PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 29/00

A2

(11) International Publication Number: **WO 98/58478**

(43) International Publication Date: 23 December 1998 (23.12.98)

(21) International Application Number: PCT/CA98/00573

(22) International Filing Date: 10 June 1998 (10.06.98)

(30) Priority Data:
2,207,746 13 June 1997 (13.06.97) CA

(71) Applicant *(for all designated States except US)*: IRONSIDE TECHNOLOGIES INC. [CA/CA]; 111 Granton Drive, Richmond Hill, Ontario L4B 1L5 (CA).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: SIKS, Andrew, R. [CA/CA]; 159 Fred Varley Drive, Unionville, Ontario L3R 1T6 (CA).

(74) Agent: DANIELS, Kent, J.; c/o Stikeman, Elliott, Barristers & Solicitors, Suite 914, 50 O'Connor Street, Ottawa, Ontario K1P 6L2 (CA).

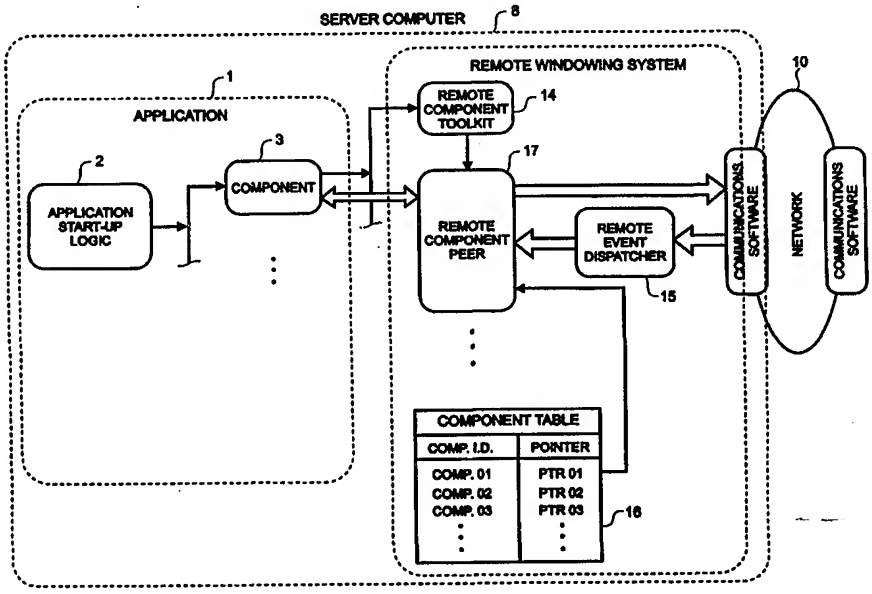
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD OF MANIPULATING SOFTWARE COMPONENTS THROUGH A NETWORK WITH ENHANCED PERFORMANCE AND REDUCED NETWORK TRAFFIC

(57) Abstract

A method of manipulating components through a network with enhan ced performance and reduced network traffic includes providing a proxy application on a client computer and a remote windowing system on a server computer. The proxy application emulates, on the basis of instruction codes received from the server computer, the components of an application running on the server computer. The remote windowing system emulates, on the bases of activity packets received from the client computer, data input and user–initiated events provided by the windowing system of the client computer. In operation, when a user–initiated event is passed to the proxy application by the windowing system of the client computer, the proxy application encodes event data indicative of the event, and transmits the encoded event data to the server computer as an activity packet. Upon receipt of an activity packet, the remote windowing system in the server computer decodes the event data, and passes the event data to a selected component of the application for processing. Component changes resulting from processing of th e event by the application are encoded by the remote windowing system and transmitted as instruction codes to the client system. Upon receipt of the instruction codes, the proxy application renders the component changes on the client computer. As a result, a user of the client computer is able to utilize the application as if it were running locally on the client computer, instead of running on a remote server computer. Network traffic between the client and server computers is reduced to encoded activity packets and instruction codes, which are small and can be transmittedquickly.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

TITLE

Method of Manipulating Software Components Through a Network With Enhanced Performance And Reduced Network Traffic

SUMMARY OF THE INVENTION

This invention relates to a method of manipulating software components through a network with enhanced performance and reduced network traffic, and in particular to a method of managing windowing components through a network such as the Internet.

For ease of understanding the present invention, the following definitions will be used throughout the remainder of the specification:

Windowing system: An object-oriented program designed to implement a Graphical User Interface (GUI) on a computer. The windowing system (such as, for example, Windows, Windows 95, OS/2, X-Windows – all trademarks) typically controls the display of information on a display monitor, and detects user activities through a keyboard and pointer devices. Data indicative of detected user activities are passed by the windowing system to a program for processing.

Application: A program designed to perform a predetermined set of tasks. The application receives user-input data via the windowing system, and passes data to the windowing system for rendering on a display monitor to be viewed by a user.

Component: An instance of an object created by operation of an application, typically to implement a functional feature of the application. Typically, each component contains data and methods which control the rendering of the component on the display screen and provide the component's functionality. For example, in order to allow the user to send data to a printer, an application will provide the user with a "Print" button which is displayed on the monitor screen by the windowing system. The user must then "click" on the displayed image of the print button to effect the desired function. The "Print" button is a component created during runtime of the application as an instance of a "button" object. The component contains data defining, for example the size, shape, color and location of the image to be rendered on the monitor), and methods defining the functionality of the component (i.e. what happens when the user "clicks" on the image of the button as it is rendered on the display monitor by the windowing system).

Event: An object that encapsulates user activities that have been detected by a windowing system. For example, as a pointer device (e.g. a mouse) is moved by a user,

the windowing system generates a series of events representing the changing coordinates of the pointer. If the user presses a key on the keyboard, the windowing system generates an event indicating the key that was pressed. Events are passed by the windowing system to the affected component, for handling in accordance with the component's methods.

Significant Event: An event which is likely to advance an application, as opposed to those events that will not advance the application. For example, when a character is keyed into a text field, a "key press" event is generated by the windowing system and passed to the text field component. However, at that point, the user may not have finished keying, so consequently this "key press" event is not significant. On the other hand, when the user presses the "Enter" key, the "key press" event generated by the windowing system and passed to the text field is significant, because is it possible that the user typed some text into the text field prior to the "key press" event, and this new text will have to be processed by the application.

Recent developments in computer technology have given rise to a dramatic increase in the use of networks to facilitate communication between computers. In particular, the Internet has experienced explosive growth, both in terms of the number of users, and the volume of data traffic. In recent years, increases in the power of personal computer (PC) technology, and corresponding increases in the data-processing power of software, have exceeded increases in the bandwidth of the network backbone. As a result, the functionality of applications designed to operate through networks are frequently limited more by the data transmission speed of the underlying network than by the capacity of the computers themselves. In effect, the network backbone, while providing the means of communication between computers, also presents the single most significant impediment to that communication. The use of so-called "Fire-wall" computers, which are typically employed to enhance security and prevent unauthorized access to computers connected to the network, further compounds this problem, as the operation of the "Fire-wall" imposes its own restrictions on data through-put.

Against this background, it has long been desirable to provide a method by which applications can operate through a network, with minimum data flow.

Advances in PC-based software since the late 1980's have been accompanied by a shift to "Object-Oriented-Programming" (OOP), which is characterized by the organization of data structures and program architecture into "Objects". As is well known to those skilled in the art, the use of OOP allows the creation of highly sophisticated

programs, with a greatly simplified program structure. A recent development of OOP is the introduction of programming languages that allow applications to run on computers using virtually any operating system (O/S), such as, for example Windows (trademark), Unix (trademark), OS/2 (trademark). With respect to the user-interface, these so-called "platform-independent" languages, such as, for example, JAVA (tradename), typically operate as described below.

During operation of an application, various "windowing components" (such as windows, panels, text fields, buttons etc.) are created and manipulated. Each component is an instance of a corresponding class of object, and includes its own data and methods. Each component also includes the definition of relationships between the component being created and other previously created components, in the manner known in the art. However, the actual implementation of a component (that is, the fine details of its screen rendition and precise manner of interaction with the user) is dependent on the operation of the particular computer system's O/S. Accordingly, in a platform-independent application, the creation of a component also includes the creation of a "component-peer", which interacts with the specific O/S to perform the required operations. For example, if an application requires that a "panel" component be created and displayed on the screen, the following steps take place. First, a panel component is created as an instance of a panel class of object, and stored in memory. A panel component-peer, which is specific to the O/S of the computer, is then created (by the panel component) and stored in memory in association with the component. When the application calls the "display" method of the panel component, the panel component calls the "display" method of its associated panel component-peer; which then interacts with the O/S to effect the display of the component on the monitor screen.

This multi-step process allows for the creation of a set of low-level windowing objects (e.g. windows, panels, buttons, text boxes etc.) for each O/S, so that both the methods of object, and the O/S are rendered transparent to the application developer. As a result, applications can be developed, using these objects, without regard to the underlying code of each object or the peculiarities of each O/S, as these details have been previously defined by the developer of the objects themselves. The application developer is therefore free to develop and implement applications with a high degree of confidence that the application can operate in essentially the same manner on every computer, regardless of the O/S being used.

Unlike conventional programs, an Object Oriented program does not have a "main program" – "sub-program" structure, in which the main program governs the control-flow of the program, and which inherently imposes restrictions upon the actions which can be taken by the user as the program operates. Instead, the operation of an Object Oriented program is furnished by means of executing the methods (procedures) of each of the various components, and interactions between the components in response to "events" initiated by the user. Thus action of the program is heavily dependent on what the user wants to do at any particular time. A "main program" or "Start-up Logic" is commonly used to define global parameters and to govern the initial creation and display of components during the start-up of the application.

In principle, this type of operation can be effected over a network, such as the Internet, between a server computer, and a client computer. One method that has been employed for this purpose involves an arrangement in which the application runs on the server computer. Components, and their corresponding component-peers are created and maintained on the server, and an image of the resulting screen display is transmitted through the network to the client computer and displayed on the screen. User inputs are transmitted by the client computer through the network to the server, and utilized in the continued operation of the application. This arrangement has a number of disadvantages. In particular, the transmission of an image of the screen display to the client computer involves the transmission of a large volume of data, so that delays in transmission can severely degrade the quality of the image displayed on the client computer. Furthermore, communication between the server and client computers is virtually continuous, which will not be permitted by many firewall systems.

An alternative method is to download the entire application from the server, so that it can be run locally on the client computer. This permits a high performance operation on the client computer, and reduces network traffic while the application is running. However, substantial (frequently unacceptable) delays in starting the application can be encountered, depending on the amount of time required to transmit the entire application through the network. Furthermore, once the application is downloaded to the client computer, the owner of the application looses control over how that application is used. For example, "pirate" copies of the application could be made and distributed in violation of the application owner's copyright.

It is therefore an object of the present invention to provide a method of operating applications through a network which avoids the above-noted deficiencies of the prior art.

It is a further object of the present invention to provide a method of operating applications through a network which substantially renders the network transparent to the application developer.

More particularly, it is a further object of the present invention to provide a method by which an application can manipulate components on a client computer through a network, while maintaining security of the application itself, and minimizing network traffic.

A further object of the invention is to provide a method by which an application can manipulate components on a client computer through a network having one or more intervening "Fire-wall" systems.

Accordingly, the present invention provides, in a network comprising at least one server computer and at least one client computer connected via a communications media, the client computer having a conventional windowing system capable of rendering components of an application on a monitor of the client computer, and further capable of passing user-initiated events to components of an application running on the client computer, a method of manipulating components of an application running on the server computer. The method comprises monitoring the windowing system of the client computer, and, upon receipt of a user initiated event. Data indicative of the user initiated event to is encoded to create an activity packet, the amount of data of the activity packet typically being smaller than that of the data indicative of the event. The activity packet is transmitted to the server computer. When received in the server computer, the activity packet is decoded to obtain event data indicative of the user initiated event. The event data is passed to the application running on the server computer to process the event. The application running on the server computer is monitored, and, upon receipt of component change data indicative of changes in a component, component change data indicative of the component change is transmitted to the client computer. When received in the client computer, the component change data is passed to components running on the client computer for rendering on the monitor of the client computer.

A further aspect of the invention provides a method of manipulating components on a remote computer via a network, the method comprises receiving component change data concerning changes in a component from an application running on a first computer. Encoded data indicative of the component change data is transmitted to the remote computer. When received by the remote computer, the encoded data is decoded to obtain component change data concerning desired component changes. Finally, the component

change data is passed to components running on the remote computer to render the desired component changes.

A further aspect of the present invention provides, in a network comprising at least one server computer and at least one client computer connected via a communications media, a method of starting an application for use by a user of a client computer. The method comprises preliminarily storing the application, a proxy application and an enabler program on a server computer. The enabler program is run continuously on the server computer to continuously monitor the communication media for a network connection attempt initiated from a client computer. The proxy application is transmitted from the server computer to a client computer via the network and initialized on the client computer. By means of the proxy application, a network connection with the enabler program on the server computer is established. The application on the server computer is initialized in response to establishment of a network connection with the enabler program. On the server computer, components are created in accordance with start-up logic of the application. A respective remote component peer for each component is also created on the server computer. By means of each remote component peer, component creation instructions are transmitted to the proxy application running on the client computer. Finally, a respective client component corresponding to each component created on the server computer is rendered on the client computer in accordance with the component creation instructions transmitted by the remote component peers.

A further aspect of the present invention provides a method of manipulating components through a network. The method comprises receiving an event from a windowing system of a client computer, the event being initiated by a user of the client computer. Event data indicative of the user initiated event is encoded to create an activity packet, and transmitted to a server computer through the network. When received in the server computer, the activity packet is decoded to obtain the event data indicative of the user initiated event. One or more methods of a component on the server computer are called in accordance with the event data. Component change data indicative of changes in a component running on the server computer are received and transmitted as component change data to the client computer. When received in the client computer, one or more methods of a component running on the client computer are called in accordance with the component change data.

A still further aspect of the present invention provides, in a network comprising a server computer and a client computer connected by a network media allowing two-way

transmission of data between the server and client computers, and wherein each computer includes a conventional windowing system capable of facilitating interaction between components of an application running on each computer and a respective user of each computer, a method for manipulating components of an application running on the server computer in response to inputs provided by a user of the client computer, with minimum network traffic. The method comprises providing a proxy application on the client computer capable of emulating, on the basis of instructions codes received from the server computer, components of the application running on the server computer. The proxy application is further capable of receiving data and user-initiated events from the windowing system of the client computer, and transmitting an activity packet indicative of the data and user-initiated events input to the server computer. A remote windowing system is provided on the server computer capable emulating, on the basis of codes received from the client computer, data and user-initiated events received from the windowing system of the client computer. The remote windowing system is further capable of receiving data indicative of changes in components of the application on the server computer, and transmitting instruction codes indicative of such component changes to the client computer. By this means, components of an application running on the server computer can be manipulated on the basis of data and user-initiated events provided by a user of the client computer, and network traffic is limited to transmission of activity packets indicative of such data and user-initiated events from the client computer to the server computer, and component change data indicative of component changes from the server computer to the client computer.

A further aspect of the present invention provides, in a network comprising at least one server computer and a client computer connected for communication via a network media, a method of initializing an application on a server computer for use by a user of the client computer. The method comprises providing and continuously running an enabler program on a server computer, the enabler program being capable of monitoring the network for, and accepting, network connection requests from the client computer. An object class definition of a proxy application is retrieved into the client computer and initialized. Using the proxy application, a network connection with the enabler program running on the server computer is established. When the enabler program on the server accepts a network connection from the proxy application, the enabler program is used to create and initializing an instance of the application's startup logic on the server. Using the application's startup logic, components are created consistent with the application's intended form of operation. As each component is created, a respective remote

component peer is created on the server computer. Using the remote component peer, component creation instructions are transmitted to the proxy application on the client computer. Finally, the component manager is used to create a client component on the client computer on the basis of the component creation instructions received from the server computer, the client component corresponding to and emulating a respective component on the server computer.

A still further aspect of the present invention provides, in a network comprising a server computer and a client computer connected by a network media allowing two-way transmission of data between the server and client computers, and wherein each computer includes a conventional windowing system capable of facilitating interaction between components of an application running on each computer and a respective user of each computer, a system for manipulating components of an application running on the server computer in response to inputs provided by a user of the client computer, with minimum network traffic. The system comprises a proxy application on the client computer, the proxy application being capable of emulating, on the basis of codes received from the server computer, components of the application running on the server computer. The proxy application is further capable of receiving data and user-initiated events from the windowing system of the client computer, and transmitting an activity packet indicative of data and user-initiated events to the server computer. A remote windowing system on the server computer, the remote windowing system being capable emulating, on the basis of codes received from the client computer, data and user-initiated events received from the windowing system of the client computer. The remote windowing system is further capable of receiving data indicative of changes in components of the application on the server computer, and transmitting instructions codes indicative of such component changes to the client computer.

Preferably, the proxy application comprises at least one client component capable of interacting with the windowing system of the client computer in a conventional manner to facilitate rendering of each client component and reception of data and user-initiated events. A component manager is capable of instantiating and manipulating each client component in response to instruction codes received from the server computer. An event handler is responsive to each client component and capable of receiving event data indicative of a user initiated event received by a client component, and transmitting the event data to the server computer as an activity packet. A client component table is provided containing component identifier and address information for each client component. The client component table provides a look-up table whereby the component

manager can select a component on the basis of a component ID received from the server computer. Finally, a component painter is capable of interacting with each client component, and the windowing system on the client computer, to control the rendering of "paintable" components on a monitor of the client computer.

Preferably, the event handler comprises an event filter responsive to each client component for determining whether or not a user-initiated event is significant. An event transmitter is responsive to the event filter for encoding and transmitting to the server computer event data as an activity packet. A change transmitter is responsive to the event filter for encoding and transmitting to the server computer induced component change data as an activity packet.

When an application is started on the server computer, a proxy-application is transmitted to and started on the client computer. The proxy-application receives instruction codes and data concerning the creation of components and changes in the state of each component from the server computer. In response to the received instruction codes and data, the proxy-application creates components and changes the state of components on the client computer. In effect, the proxy-application reproduces on the client computer the component changes occurring on the server computer as a result of operation of the application. The proxy-application is further designed to receive events generated by the windowing system of the client computer, and transmit instruction codes and data concerning such events to the server computer.

The proxy-application operates in the client computer, and mimics the operation of the application. Like a conventional application, the proxy-application receives events generated by the windowing system, and creates and manipulates components in response to those events. However, rather than handling the events and determining component changes locally, the proxy-application transmits encoded events and component data to the server computer, and then receives instruction codes and data concerning component changes from the server computer. Since the interface between the proxy-application and the windowing system on the client computer is identical to that which exists between a conventional application and windowing system, the existence of the network between the client computer and the application on the server computer is rendered transparent to the user of the client computer.

At the same time, a remote windowing system is activated and maintained in the server computer. The remote windowing system is designed to receive encoded events and component data transmitted by the proxy-application on the client computer. In

response to the received event data, the remote windowing system triggers appropriate component methods so as to cause the affected components to handle the events in an appropriate manner. In effect, the remote windowing system reproduces on the server computer the events generated by the windowing system on the client computer. The remote windowing system is further capable of receiving component operations from the application on the server computer, and transmitting to the client computer instruction codes and data concerning the creation of components and changes in the state of each component.

The remote windowing system operates in the server computer, and mimics the operation of the windowing system on the client computer. Like a conventional windowing system, the remote windowing system generates events based on actions of the user, and receives component operations concerning component changes resulting from operation of the application. However, rather than responding to a user of the server computer and effecting component changes on the server computer, the remote windowing system receives encoded events and component data from the client computer, and then transmits instruction codes and data concerning component changes to the client computer. Since the interface between the application and the remote windowing system is identical to that which exists between an application and a windowing system in a conventional discrete system, the existence of the network between the application on the server computer and the windowing system on the client computer is rendered transparent to the application.

In an embodiment of the invention, the remote windowing system comprises a remote component toolkit (RCT), a Remote Event Dispatcher, a Component table, and a plurality of remote-Component Peers (RCP). The RCT receives, from the client computer, instruction codes and data concerning events generated by the system thread, and triggers actions of the application in response to those events. Each RCP interfaces with a respective component of the application running on the server computer in a substantially conventional manner, and transmits instruction codes and data concerning component changes to the client computer.

In an embodiment of the invention, the proxy-application comprises Component Management Software (CMS) and Event Transmission Software (ETS). The CMS receives from the server computer instruction codes and data concerning component changes resulting from operation of the application, and causes corresponding component

changes on the client computer. The ETS receives events generated by the windowing system, and transmits encoded events and related data to the server computer.

Preferably, the proxy-application is capable of discriminating between a significant event and an insignificant event, whereby only data concerning significant events is transmitted by the ETS to the server computer. By this means, the amount of network traffic and server load can be reduced by eliminating the transmission of insignificant events.

Preferably, the proxy-application is further capable of accumulating information concerning changes in the data stored in one or more components until a significant event occurs. When a significant event does occur, the accumulated information is transmitted by the ETS along with the instruction codes concerning the significant event, as a single transmission. By this means, the number of data transmissions can be reduced.

Preferably, the remote windowing system transmits to the client computer instruction codes and data concerning component changes resulting from operation of the application only in response to receipt of an event from the client computer. By this means, the exchange of all instruction codes and data between the server computer and the client computer occurs in the form of a sequence of Event/Response cycles, all of which are initiated by the client computer. Event/Response cycles of this nature are typically accepted by intervening Fire wall systems, so that operation of Fire Wall systems will not interfere with the communications between the server computer and the client computer. Furthermore, the remote windowing system and the proxy-application are preferably capable of transmitting and receiving instruction codes and data formatted as either binary data, or an HTTP/HTML request and response. By this means, a data transmission format acceptable to any intervening Fire Wall systems can be selected and used during the course of operation of an application. As a result, the existence of an intervening Fire wall system is rendered substantially transparent, without circumventing the Fire Wall security features or otherwise violating system security.

Advantages of the present invention include:

1. In most cases, the size of the proxy-application will be significantly smaller than the application itself. As a result, the time required to download the proxy-application to the client computer is proportionately less than would be the case if the application were to be downloaded.

2. Because the proxy-application only contains code necessary to replicate component changes caused by the application, it follows that the same proxy-application can be used for a great many different applications. Thus the time required for a user (on a client computer) to start an application (which includes downloading the proxy-application) is constant, and independent of the size of the application.

3. With the use of the present invention, since applications are run on the server and not transmitted to the client, the application software cannot be examined, reverse engineered or altered to achieve results other than those intended by the authors.

4. By using the present invention for applications that display or use server-resident or server-accessible data, there is no need to devise protocols or methods for the transmission of data to the client. Applications that would otherwise be 2-tier or 3-tier applications can be written as 1-tier or 2 tier applications respectively. This substantially reduces application development time, maintenance time, and the necessary skill levels of developers, without sacrificing the performance or quality of the application.

5. When the present invention is used, network traffic is spread out over time thus eliminating the heavy bursts required to download applications. In many cases, for a single use of an application, the total network traffic is less than that which would conventionally be needed to merely download the application.

A specific embodiment of the present invention provides a process whereby a Java program running on one computer (the server computer) creates and manipulates Java Components on one or more remote computers (client computers), through a network, and receives Events from these Components. The Java Components on each client computer run independently of Java Components on other client computers, while being remotely manipulated by the server computer. By this means, the need to transmit large numbers of Java class files to the client computer is eliminated, thereby significantly reducing delays perceived by a user of the client computer. In this case, A Java application on the server computer, in a conventional manner. When the getToolKit() method of a Java Component without a peer or parent is called, an instance of a Remote Component Toolkit (RCT) is returned. When the addNotify() method of any non-abstract method is called, the appropriate "create" method of the PCT is called. The create method

instantiates a Remote Component Peer (RCP) of the correct type and returns it to the Java Component. The RCP communicates with the Component Management Software (CMS) on the Client computer. The RCP transmits component related instructions to the CMS. The CMS converts these instructions into component method calls that result in the creation and manipulation of components on the client computer. When an Event is generated on the client computer (by action of the user) and delivered to the postEvent() method of a component, the Event, of significant, is codified and transmitted to a Remote Event Dispatcher of the Server computer. The Remote Event Dispatcher re-creates the Event on the server computer and calls the postEvent() method of the corresponding Java Component on the server computer.

The method of the invention comprises the steps of: Receiving, from an application, a method call for a selected component; Transmitting to the client computer information indicative of the selected component, the desired method call and its parameters; In the client computer, receiving the information from the server computer; Decoding the information to extract the ID of the selected component, the desired method and its parameters; and Issuing the method call to the selected component on the client computer.

The method of the invention comprises the steps of: Receiving an event from a windowing system; Analyzing the event to determine whether or not the event is significant; when the result of this step is "yes", determining whether or not the data in any component has changed; When the result of this step is "yes", transmitting to the server computer information indicative of the component and its contents; In the server computer receiving the information from the client computer; Decoding the information to extract the ID of the selected component and its contents; and storing the contents in memory for subsequent access by the application. Transmitting to the server computer "event" information indicative of the event and its associated component; In the server computer receiving the information from the client computer; Decoding the information to extract the ID of the selected component and the event; and passing the event to the selected component in accordance with the methods of the selected component.

BRIEF DESCRIPTION OF THE DRAWINGS

Further features and advantages of the present invention will become apparent from the following detailed description, taken in combination with the appended drawings, in which:

Figure 1 is a simplified schematic illustration of a conventional Object Oriented application running on a discrete system;

Figure 2 is a simplified schematic illustration of a conventional Object Oriented application running between a server computer and a client computer through a network;

Figure 3 is a schematic illustration showing the server-side of an Object Oriented application running on a server computer and communicating with a client computer through a network in accordance with the present invention;

Figure 4 is a schematic illustration showing the client-side of an Object Oriented application running on a client computer and communicating with a server computer through a network in accordance with the present invention;

Figure 5 is a schematic illustration showing the Event-Handling Software of the proxy-application of Figure 4;

Figures 6a and 6b show a flow-chart illustrate the general sequence of steps in the method of the present invention;

Figures 7a-7b are network transmission diagrams illustrating the transmission of graphical information data from a server computer to a client computer;

Figure 8 is a schematic illustration of a display monitor screen of a client computer which is connected with a server computer running an example application in accordance with the present invention; and

Figures 9a-9f are network transmission diagrams illustrating the transmission of data between the client computer and server computer in the example of Figure 5.

DETAILED DESCRIPTION

For ease of understanding the present invention, a detailed description thereof is preceded by the following description of the prior art with reference to Figures 1 and 2.

A conventional Object Oriented application 1 running on a discrete computer utilizes the general structure illustrated, in simplified form in Figure 1. The application 1 comprises application start-up logic 2, and one or more components or extended components 3a-c. The application 1 makes use of a windowing system 4 that contains a windowing toolkit 5 capable of creating component peers 6a-c corresponding to each component 3.

The application start-up logic 2 controls the initialization, creation, and display of components3 during the start-up of the application 1. Subsequent creation and manipulation of components 3 is triggered by other components 3 in response to events received from the windowing system 7. The windowing toolkit 5 creates component peers 6 for each type of component 3 utilized by the application 1, and is called upon for this purpose by each component 3 as it is created.

Changes in the status of each component 3, during operation of the application 1, are passed by the component 3 to its associated component peer 6, which communicates the component changes to the operating system 7 to effect any necessary operations (for example to cause the monitor display to change). In addition, events are passed by the operating system 7 to the affected component peer 6, which then passes the event on to its associated component 3 for processing in accordance with the component's methods.

This basic structure can be adapted to work through a network. In this case, two alternative methods are typically used. The first is to simply download the application program to the client computer having the correct operating environment. In this case, when the application program is subsequently started, the operating structure described above in connection with Figure 1 is set up. This arrangement allows the application to run on the client computer as a discrete system, thereby avoiding any performance difficulties due to transmission speed of the network. However, the amount of time required to download the entire application can be substantial. For example, a typical application program may be on the order of 700,000 bytes in size. Using a typical 28.8kbs modem, such a file (if not compressed) can be expected to take at least 243 seconds to download, assuming that the network is capable of transmitting information at the maximum data transmission speed of the modem, (which in practice is seldom the case). Such a long time delay is unacceptable for commercial applications to be viable.

Furthermore, when the application is downloaded to a client computer, the owner of the application looses control of the use of the application program, and any proprietary information and/or methods contained within it. For example, the user of the client computer could easily make and distribute unauthorized copies of the application. Additionally, the user of the client computer could attempt to "reverse-engineer" the application to determine proprietary methods developed by the owner of the application.

For both of the above reasons, the downloading of large applications to client computers is undesirable. Accordingly, in many cases, application developers often elect to use an alternative method, illustrated in Figure 2. In this case, the application software

is divided into two parts or "tiers". One part runs on the server computer 8 while the other part runs on the client computer 9. Only the client-side part creates window components that make use of the windowing system on the client computer 9. The responsibilities of the client-side part are typically limited to user interaction while the server side part handles the details of data access, computations , or "business" logic. At certain logical points, the client-side part transmits operational requests and user inputs to the server-side part, which performs the required operations and transmits the resultant information to the client-side for display to the user.

This arrangement has the advantages that the user of the client computer 9 experiences less delay while the application 1 is downloaded, and security of the application 1 is improved. However, the design of the 2-tier application is rendered substantially more complicated, because the application designer must make potentially difficult decisions with respect to partitioning the overall logic and must develop the protocols for bi-directional communications at the partition points. The testing and trouble-shooting of such 2-tier systems also becomes increasingly more complex. Finally, although the amount of application code needed on the client computer may be reduced, it can still be unacceptably large for transmission using conventional modems.

The present invention employs a combination of application-specific and generic (i.e application-independent) software which is initialized and stored into memory of the server computer 8 and the client computer 9 when an application is started. For simplicity, the structure and function of the software loaded onto each of the two systems will be described in general terms, with reference to Figures 3 - 5. The method by which this structure transmits information between the server and client computers will then be described with reference to Figures 6a-6b, followed by a description of the method by which the entire system is initialized and loaded. Finally, specific examples of the operation of the present invention will be described with reference to Figures 7a-b, 8, and 9a-f.

Referring now to Figures 3 and 4, an object oriented application running in accordance with the present invention comprises an application 1 program and a remote windowing system 11 on the server computer 8, and a proxy-application 12 and client windowing system 13 running on the client computer 9.

The application 1 includes application start-up logic 2, and one or more components 3 (only one is shown in Figure 3), each of which are structured and function

in a conventional manner as described above in connection with Figure 1, and thus will not be described in further detail here.

The remote windowing system 11 operates in the server computer 8 and interacts with the components 3 of the application program 1 in a manner substantially identical to the windowing system 4 in a conventional discrete system (as described above in connection with Figure 1). In use, the remote windowing system 11 serves to emulate the client windowing system 13 on the client computer 9 in such a way as to render the intervening network 10 transparent to the application 1. The remote windowing system 11 comprises a remote component toolkit 14, a remote event dispatcher 15, a component table 16, and a respective remote component peer 17 corresponding to each component 3 of the application 1. Each remote component peer 17 is designed to interface with a respective component 3, and to transmit instruction codes and data concerning changes in the state of its respective component 3 to the proxy-application 12 running in the client computer 9. The remote event dispatcher 15 is designed to receive, from the proxy-application 12, instruction codes and data concerning events generated by the client windowing system 13 on the client computer 9. The format and origin of these instruction codes and data will be described in greater detail below. Upon receipt of events and data from the proxy-application 12, the remote event dispatcher 15 uses the component table 16 to locate and call the affected remote component peer 17, which in turn call methods of its associated component 3 in order to trigger that component's methods in substantially the same manner as is typically performed by conventional component peers upon receipt of events from the windowing system in a conventional discrete system.

Like a conventional windowing toolkit, the remote component toolkit 14 creates instances of the appropriate type of remote component peers 17 when called to do so by a component 3. The object definitions for the remote component peers 17 are selected from a plurality of previously defined object definitions compatible with the O/S and communications software operating on the client computer 9, so that component instructions and data concerning changes in components can be transmitted efficiently to the client computer 9.

In accordance with the present invention, upon receipt of a component change from its associated component 3, the remote component peer 17 transmits an encoded component instruction to the client computer 9 through the network 10. The specific contents of the component instruction will normally vary in accordance with the associated component's object definition, but always includes sufficient information for the component

change to be fully replicated in the client computer 10. Thus if the text in a text-field component is being changed, for example, the encoded component instruction transmitted to the client computer will contain: a code indicating that text-field data is being transmitted; the component identifier of the affected component; the length of the data; and the data itself.

Referring now to Figure 4, the proxy application 12 runs in the client computer 9, and interacts with the client windowing system 13 in a manner substantially identical to the interaction between a conventional application and a windowing system of a discrete computer as described above in connection with Figure 1. Conveniently, the proxy application 12 comprises a component manager 18, an event handler 19, a respective client component 20 corresponding to each component 3 of the application program 1, a component painter 21, and a client component table 22.

The component manager 18 is designed to receive encoded component instructions transmitted by the remote windowing system 11 on the server computer 8. Upon receipt of the component instructions, the component manager 18 decodes the instructions, selects the affected client component 20 from the component table 22, and effects the desired changes in the selected client component 20. The component painter 21 is used in the rendering of graphic objects associated with any client component 20. The specific operations performed by the component manager 18 will normally depend on the component instructions received, but always replicates the component changes passed by the corresponding component 3 to the remote component peer 17 in the server computer 8.

The event handler 19 is designed to receive data concerning user-initiated events and component changes from each client component 20, and selects those events deemed to be significant, and then transmits an activity packet containing encoded data concerning significant events and component changes to the remote event dispatcher 15 in the server computer 8. Conveniently, the event handler 19 includes an event filter 23, a change transmitter 24, and an event transmitter 25.

The client windowing system 13 is substantially conventional, and comprises a windowing toolkit 4 which is used by each client component 20 to create its corresponding component peer 6 in a conventional manner. Interactions between each client component 20 of the proxy application 12 and the client windowing system 13 on the one hand, and between the component peers 6 and the operating system 7 of the client windowing system 13 on the other hand, are entirely conventional, and thus will be described in

further detail only with respect to discrimination between significant and insignificant events.

In general, every action taken by the user results in the generation of one or more events. For example, movement of a pointer device (e.g. a mouse, not shown) produces a stream of events corresponding to the changing coordinates of the pointer device. Pressing a button on the pointer device, or a keyboard, produce other events respectively, the specific nature of which (in terms of their effects on the component peers and components) will vary depending on numerous factors, including which component has the focus, and how the methods of that component and its corresponding component peer have been programmed to respond to that event.

It will be seen that a great many events will have no meaningful effect on an application, and can therefore be disregarded as being insignificant. The specific events which will be considered to be significant will depend largely on the operational features of the application as determined by the designer. For example, in an application (or, more precisely a panel component of an application) in which the user is able to enter information into one or more text fields, and "click" an "Ok" button to indicate that they are done, there is only one significant event; namely the "Action" event associated with the clicking of the "Ok" button.

Other events, such as those generated when the mouse is moved, or when a text field receives the focus can be deemed to be insignificant, because they do not provide useful information (in respect of the advancing state of the application). It will be noted that the receipt of text characters into a text field component can also not considered to be significant. This is due to the fact that as long as the user is entering text in a text field component, the contents of that text field is changing. The text data that is important, and which must be transmitted back to the server, is the text data in the text field when the user has finished typing. Thus it is sufficient to allow the text field component (or its corresponding component peer) to receive and accumulate changes in the text data while the user is typing, and then only transmit the final data back to the server (as a user-induced component change) upon receipt of the "Action" event corresponding to the click of the "Ok" button.

It will be apparent that other applications, employing significant numbers of components, and components of different types, will have their own specific list of events that will be deemed to be significant, depending on the operational features of the specific application in question. Furthermore, it will be seen that the activities of the client and

server computers can proceed in parallel; with the application 1 on the server computer 8 processing received events to determine component changes, while at the same time the proxy-application 12 on the client computer 9 continues to receive and process user initiated events. However, in general, the method of the present invention can be represented as a series of Event/Response cycles, in which: a user initiated (significant) event is transmitted by the proxy-application 12 to the server computer 8; the event is processed by the application 1 to yield component changes; and these component changes are then transmitted back to the proxy-application 12 for rendering on the client computer 9. In this context, the general steps in the method of the present invention will now be described with reference to Figures 6a and 6b.

Within a client windowing system 13 compatible with present invention, all events (significant or otherwise) are passed from the operating system 7 to the component peer 6, which passes the event on to its associated client component 20 in a conventional manner at step S1. The client component 20 automatically passes the event (again without regard to whether the event is significant) to the event filter 23, which is programmed to discriminate between significant and insignificant events (step S2), depending on the operational requirements of the application 1 as determined by the application designer. Insignificant events are checked to determine whether or not they effect component changes (at step S3) and either accumulated as induced component changes at step S4 (as briefly described above, and in further detail in the examples below), or ignored (step S5). In contrast, significant events are passed to the event transmitter 25 for encoding and transmission to the remote event dispatcher 15 in the server computer 8.

The event transmitter 25 receives data concerning significant events from the event filter 23; determines whether any induced component changes exist to be transmitted to the server 8 (step S6); if necessary, triggers the change transmitter 24 to transmit data concerning user-induced component changes; and then transmits to the server computer 8 instruction codes and data concerning those events and induced component changes. Conveniently, instruction codes and data concerning induced component changes, and instruction codes and data concerning events will be transmitted sequentially, as separate logical blocks. Thus if it is determined at step S6 that induced component changes exist to be transmitted to the server 8, the event transmitter 25 triggers the change transmitter 24 to assemble an activity packet containing inducted component changes (Step S7) which is then transmitted to the server computer 8 (Step S8). The format and specific contents of the activity packet containing instruction codes

and data concerning induced component changes will normally vary in accordance with the associated component's object definition, but always includes sufficient information for the induced component change to be fully replicated in the server computer. Thus upon receipt of an "Action" event corresponding to the click of an "Ok" button (following the example above), the instruction codes and data concerning induced component changes transmitted to the server computer 8 will contain: a code indicating that text-field data is being transmitted; the component identifier of the affected component; the length of the text data being transmitted; and the text data itself.

The activity packet containing induced component changes is received by the remote event dispatcher 15 in the server computer 8 (Step S9), the activity packet is decoded to extract the identifier of the affected component 3 and the component changes involved (Step S10), and the component changes stored with the selected component for later use by the application 1 (Step S11).

Once the event handler 19 has completed transmitting the activity packet containing induced component changes to the server computer 8, or when the event transmitter 25 determines (at Step S6) that no induced component changes exist to be transmitted, then the event transmitter 25 constructs an activity packet (at Step S12) for the event passed from the event filter 22 at step S2. This activity packet is then transmitted to the server computer 8 (Step S13). As with the activity packets containing induced component changes, activity packets containing instruction codes and data concerning significant events will normally vary in accordance with the associated component's object definition, but always include sufficient information for the event to be fully replicated in the server computer 8. Thus upon receipt of an "Action" event corresponding to the click of an "Ok" button, and following transmission of induced component changes as described above, (again following the example above), the instruction codes and data concerning the event transmitted to the server computer will contain: a code indicating that event data is being transmitted; the component identifier of the affected component; the event identifier; and any arguments which accompany that event.

The activity packet containing instruction codes and data concerning significant events is received by the remote event dispatcher 15 in the server computer 8 (Step S14), decoded to extract the identifier of the affected component 3 and details of the event (Step S15), and the event passed to the selected component 3 (or remote component peer 17) at step S16. The event is then processed by the component 3 (Step S17) in accordance

with the component's methods in a manner identical to that following receipt of an event in a conventional discrete computer. However, any component changes resulting from processing of the event are not rendered on the server computer 8 as would conventionally be the case. According to the present invention, changes in any component 3 (resulting from processing of an event) are passed to the corresponding remote component peer 17 which transmits instruction codes and data concerning the involved component changes to the component manager 18 in the client computer 9 (Step S18).

Upon receipt of instruction codes and data concerning the component changes from a remote component peer 17 (Step S19), the component manager 18 selects the affected client component 20 and passes the instruction codes and data to the affected client component 20 (Step S20) to be processed and rendered in accordance with the client component's methods and the client windowing system 13.

In summary, in the server computer 8: the present invention replaces the conventional windowing system 4 (comprising a windowing toolkit 5 and component peers 6) with a remote windowing system 11 (comprising a remote component toolkit 14, remote component peers 17, component table 16, and a remote event dispatcher 15). By this means, the existence of the network 10 is effectively rendered transparent to the application program 1 and components 3, which therefore can issue component operations, and receive events as if the client windowing system 13 was running locally on the server computer 8.

Similarly, in the client computer 9: the present invention replaces the application with a proxy application 12 comprising the component manager 18, event handler 19, one or more client components 20, a component painter 21 and a component table 22. By this means, the existence of the network 10 is effectively rendered transparent to the user and client windowing system 13, which therefore receives component operations, and dispatches events, as if the application 1 was resident and running in the client computer 9.

In effect the present invention intervenes between the application program and components resident in the server, and the windowing system in the client computer, to render the network transparent to both the application and the windowing system.

Examples

In the following, the present invention will be illustrated by means of specific examples, each of which involve applications written in Java (trade name). In each example, the network connecting the server and client computers is the Internet, and in particular the World-Wide-Web, and the use of conventional TCP/IP and HTTP protocols, sockets, and URL's is assumed. Similarly, in each case, conventional operating systems, such as, for example Windows (Trademark), Windows NT (Trademark), and Unix (Trademark), and Java-enabled browser programs such as Netscape Navigator (Trademark) and Microsoft Internet Explorer (Trademark) are assumed to be operating in the server and client computers. Thus it will be apparent that the embodiments of the invention described in the following examples can be readily implemented in conventional Intranets (which also typically employ TCP/IP and HTTP protocols), for example. Those skilled in the art, however, will also recognize that the present invention is not limited to Java applications, or any particular network architecture or communications protocol. Indeed, the skilled artisan will recognize that the methods of the present invention can be employed successfully in any environment which satisfies the following conditions:

1. The operating systems running in both of the client and server computers can support event-driven object oriented applications.
2. The network communications software is capable of responding to application components by transmitting data through the network.
3. The network communications software is capable of receiving commands and data from the network and passing those commands and data to application components.
4. Application components in the client computer are capable of acting on commands and data received through the network (via the communications software), and events initiated by the user of the client computer.

Example 1: Conventional Initialization of an application

A conventional application, written in Java for use over the internet, which does not make use of the present invention, is started in the following manner.

1. The user of a conventional Java-enabled web browser (such as Netscape Navigator) may click on a link which causes the browser to retrieve, from an HTTP server, an HTML page containing an HTML "applet tag". The applet

tag names the Java class corresponding to the object class definition of the application's start-up logic, which must extend the "applet" class. On encountering the applet tag, the browser retrieves the object class definition of the application's start-up logic from the HTTP server and creates an instance of the object. The browser calls a sequence of methods of the object representing the application as is conventionally prescribed for starting any object of the applet class. This sequence includes the "init" method.

2. When the application's "init" method is called, the application proceeds to create components and extended components consistent with its intended form of operation. As additional object class definitions are required by the application. the browser retrieves them from the HTTP server.

Example 2: Initialization of an application employing the present invention.

An application, written in Java for use over the Internet, which makes use of the present invention, may be started in the following manner:

1. An "enabler" program, written in Java, is run (continuously) on a server computer 8. The enabler program listens for, and accepts, network connections.

2. The user of a browser on a client computer 9 clicks on a link which causes the browser to retrieve, from an HTTP server, an HTML page containing an HTML applet tag. The applet tag names the Java class corresponding to the object class definition of a proxy application 12, which extends the applet class. On encountering the applet tag, the browser retrieves the object class definition of the proxy application 12 from the HTTP server and creates an instance of the object. The browser then calls a sequence of methods of the object representing the proxy application 12 as is conventionally prescribed for starting any object of the applet class. This sequence includes calling the "init"' method. Note that this step is identical to step 1 of Example 1 above, for an application operating conventionally, with the exception that a proxy application 12 has replaced the start-up logic of the application itself. Notice also that the HTTP server from which the HTML page and the object class definition of the Proxy-Application 12 may or may not be the same as server computer 8 on which the enabler program, or the application program itself are run.

3. When the proxy application's "init" method is called, the proxy application 12 establishes a network connection with the enabler program running on the server computer 8 and, conveniently, detects and transmits characteristics of the client computer 9, such as, for example the type and version of the operating system and the resolution of the monitor screen, to the server computer 8.

4. When the enabler program on the server 8 accepts a network connection from the proxy application 12, it creates an instance of the application's startup logic 2 on the server 8. The enabler program ensures that the conventional windowing toolkit 5 has been replaced by a remote component toolkit 14 that is compatible with the proxy application 12 running on the client computer 9 and the detected characteristics of the client computer 9. The enabler program then calls the methods of the application's start-up logic 2 as prescribed for starting any object of the applet class.

5. When the application's "init" method is called, the application 1 proceeds to create components 3 and extended components consistent with its intended form of operation. As additional object class definitions are required by the application 1, they are available locally on the server computer 8.

6. Because the enabler program has replaced the conventional windowing toolkit 5 with a remote component toolkit 14, any component peers that are created by application components 3 are in fact remote component peers 17. The remote component peers 17 encode and transmit component creation and change instructions to the proxy application 12 running on the client computer 9 in the manner previously described. The proxy application's component manager 18, acting on these instructions, creates and renders corresponding client components 20 on the client computer 9.

Upon completion of the above initialization sequence, the user of the client computer 9 will observe the various components of the application 1 displayed on their monitor screen in substantially the same manner as if the application 1 had been initialized and was operating entirely within the client computer 9. The user is then able to make keyboard and mouse inputs to continue with utilization of the application 1, with communication of data proceeding between the server computer 8 and the client computer 9 as illustrated in the following example application.

Example 3: creation of components, and rendering of same on the client computer 9.

Figures 7a and 7b illustrate the process of creation of a component 3 on the server computer 8, through to rendering of that component on the monitor of the client computer 9. for the purposes of the present example, the application 1, remote windowing system 11, and the proxy application 12 will be assumed to have been initialized as described in example 2 above. Thus there will be at least one component 3 (corresponding to the applet panel) already in existence, which will serve as a "parent container" for the new component. Similarly, a client component 20 corresponding to the "parent container" component 3, and its component peer 6 in the client windowing system 13 will already exist.

In the present example, a new component will be created and added to the "parent container", a corresponding remote component peer 17 will be created and its identity and attributes transmitted to the client computer 9; a corresponding client component 20 is then created and added to the client component of the "parent container", which precipitates the creation of a client component peer 6 and rendering on the monitor of the client computer 9. The skilled artisan will note that, in cases where a "parent container" does not exist (such as during initial start-up of an application), suitable adjustments in the process described below can be readily made, as will be seen later in Example 4. Similarly, the skilled artisan will recognize that new components can be created by any components of an application, and thus the present invention is not limited to cases where components are created by operation of the application logic per se, as in the present example.

In general, in order for a new component 3 to be rendered on the client computer 9, all relevant attributes of the component must be communicated to and made available on the client computer 9. Clearly, a remote component peer 17 could transmit all relevant attributes each time a component 3 is created. However, since it can be observed that components typically have many attributes in common, to reduce the total number of bytes actually transmitted during the course an application's operation, the component manager 18 on the client computer 9 is provided with a set of "state" variables for attributes that may remain unchanged from one component creation to another. Accordingly, the remote component peers 17 maintain a matching set of state variables and only send changes to state variables when needed. Each such state variable change is sent to the client

computer 9 as either a lone Operation Code, or an Operation Code followed by data to be associated with the state variable.

Referring now to Figure 7a, when a new component is created by operation of application logic, the application calls various methods of the new component 3, as prescribed for starting any object of the applet class, in order to set the various attributes of the new component. The application then calls the "add" method of the parent container to add the new component to the parent container, which precipitates creation of a remote component peer 17 for the new component 3 using the remote component toolkit 14.

Following its creation, the remote component peer 17 first ensures that a number of "state" variables will be correctly set on the client computer 9. These state variables include: the parent container's component ID, the new component's visibility and enabled states, foreground and background colors, font specifications, paint instructions, and possibly others depending on the type of component. The remote component peer can obtain the parent container's component ID from the remote component peer corresponding to the parent container of the new component. The visibility and enabled states, colors and font specifications must be obtained by calling appropriate methods of the new component. To generate the paint instructions, the remote component peer creates a remote graphics object and passes it to the new component's "paint" method. The remote graphics object implements all of the methods required of a graphics object. As the various graphics drawing methods the remote graphics object are called by the "paint" method of the new component 3, the remote graphics object encodes each drawing operation into a byte sequence that is appended to a byte array. When the "paint" method finishes, the remote component peer 17 can get the byte array from the remote graphics object for transmission.

Following transmission of the state variables, the remote component peer 17 obtains a new component ID for the new component 3. The new component ID and a pointer to the remote component peer 17 are stored in the component table 16. The component ID is also stored within the remote component peer 17 itself for rapid future reference. The remote component peer 17 acquires location, size, and possibly other attributes by calling methods of the new component 3 and transmits component instructions similar to:

- Operation Code for New Component
- Component ID of the New Component

- Location (X and Y position on the screen)
- Size (width and height of the component)
- Component Type Code (i.e. for this example, a code for "Button")
- Possible additional component-specific data (for a Button this may include the label, e.g. "Start")

Following transmission of these component instructions, control returns to the application logic for continued operation of the application.

Referring now to Figure 7b, as the component manager 18 on the client computer 9 receives the state variable operation codes from the server computer 8, the appropriate state variables are set. Some state variables may be pointers to objects that are constructed using received information (the font, for example). In the case of the parent container ID, the component manager 18 converts the ID into a pointer to the referenced container client component 20 using the client component table 22. Upon receipt of the Operation Code for "New Component" and following data, the component manager 18 creates a new client component 20 of the specified type (e.g. "button") and adds the new component ID and new client component pointer into the client component table 22 for future reference. The component manager 18 then sets all attributes of the new client component by calling methods of the new client component (many of the attributes coming from state variables) and stores a pointer to the current array of paint instructions within the new client component 20 (if the component is a paintable component). Finally, the component manager 18 adds the new client component 20 to the current parent container component which causes the creation of a conventional component peer 6 in the client windowing system 13 and consequently the display of the new client component 20 on the display monitor of the client computer 9. As a consequence of this peer creation, the client windowing system 13 on the client computer 9 calls the "paint" method of the new client component (if it is "paintable"). The "paint" method calls the component painter 21, passing to it the graphics object received from the client windowing system 13 and the stored byte array of paint instructions. The component painter 21 decodes the byte away of paint instructions and calls methods of the graphics object in a sequence and manner identical to the original calls made by the "paint" method of the new component 3 on the server computer 8.

Example 4: a simple application.

An illustrative example of the present invention will now be provided with reference to Figures 8, and 9a-f. In this example, a System Performance Test Panel 26 (Figure 8) is displayed within a browser frame 27 on a display monitor 28 of the client computer 9. The System Performance Test Panel 26 includes an input text field 29, an output text field 30, and a button 31 initially labeled "Start". Each of the input and output text fields 29 and 30 are instances of a "TextField" object, which includes text data (corresponding to the contents of the respective field) and methods for reading and writing the text data. The button 31 is an instance of a "Button" object.

The System Performance Test Panel 26 is started on the client computer 9 as a result of the user of the browser reaching an HTML page that contains an applet tag referencing a proxy application 12 which precipitates the running of the appropriate application 1 on the server computer 8. The client components 20a-20e are created and displayed on the client computer monitor 28 as a result of instruction codes received from the server computer 8 during and after startup of the application 1.

In this example, the application 1 performs the following operation: when the "Start" button 31 is clicked, the application 1 changes the button label to "Stop", starts a timer, and then executes a calculation a number of times determined by the number in the Input text field 29. While the calculations are being executed, a progress bar 32 on the panel 26 is periodically re-painted to show the progress being made. Upon completion of the calculations, the timer is stopped and the total elapsed time is displayed in the output text field 30 and the button label is restored to "Start".

Figure 8 illustrates a client monitor display after initialization of the application 1 and creation of the System Performance Test Panel 24 and its elements 29-31. Thus in the state illustrated in Figure 8, the application 1 has advanced to the point where it is ready to receive input from the user. In the illustration of Figure 8, the input text field 29 contains the number "100", which for the purpose of the present example, will be assumed to have been input by the user.

Figures 9a through 9c illustrate the activities that take place leading to the state shown in Figure 8. On the client computer 9 the browser creates an instance of the applet panel representing the proxy application 12 (Step ES1). The key logical elements of the proxy application 12 (i.e. component manager 18, event handler 19, component painter 21 and startup logic) are assumed to be static methods of the proxy application 12 and hence are available with the creation of the proxy application instance. The browser then

continues in the manner prescribed for the starting of applets by setting attributes of the applet panel (Step ES2), adding the applet panel to the browser frame (Step ES3) and calling the applet panel's "init" method. During execution of the "init" method, an execution thread is started which performs the necessary start-up processing. This start-up includes the establishment of communications with the enabler program on the server computer 8 (Step ES5); initialization of the client component table 22 to contain references to the browser frame and applet panel client components 20a and 20b (Step ES6); and the transmission of "start up" information to the enabler program (Step ES7). The "start up" information can be expected to contain, at minimum, the component attributes of the applet panel that was instantiated by the browser. After transmitting the "start-up" information, the thread begins execution of the component manger 18, which starts reading from the communications connection and blocks until the arrival of data from the server computer 8.

Referring now to Figure 9b, when the enabler program running on the server computer 8 receives the "start-up" information, it must "mirror" the two components that already exist on the client computer 9 (that is, the browser frame and applet panel). To that end, the enabler program creates and "shows" a browser frame component 3a (Step ES8). By "showing" the browser frame component 3a, a browser frame remote component peer 17a for the browser frame component 3a is created by the remote component toolkit 14. The browser frame remote component peer 17 does not transmit component creation instructions to the client computer 9 because of a (implementation-specific) "Browser frame marker" associated with the browser frame object.

After mirroring the browser frame client component 20a, the enabler program creates the applet panel component 3b (Step ES9) corresponding to a target application (the identity of the target application may have been transmitted in the "start-up" information). The enabler program sets the applet panel 's attributes (Step ES10) according to the "set-up" information received from the client computer 9 and adds the applet panel component 3b to the previously created browser frame component 3a (Step ES11). Adding the applet panel component 3b to the browser frame component 3a precipitates the creation of an applet panel remote component peer 17b for the applet panel (Step ES12). The remote component peer for a panel is conveniently programmed to recognize a panel with a parent container (in this case the browser frame) having the "Browser frame marked", and consequently not transmit component creation instructions to the client computer 9. The applet panel remote component peer 17b must, however,

send, to the client computer 9, instructions to paint the applet panel. In the present example, these paint instructions are used to render the labels under the text fields 29, 30 and draw and label the progress bar area 32. To generate the paint instructions, the applet panel remote component peer 17b creates a remote graphics object that it passes to the applet panel component's paint method. The remote graphics object implements all of the methods required of a conventional graphics object. As the various graphics drawing methods of the remote graphics object are called by the "Paint" method of the applet panel component 3b, the remote graphics object encodes each drawing operation into a sequence of bytes that are appended to a byte array. When the "paint" method finishes, the applet panel remote component peer 17b extracts the byte array from the remote graphics object and sends (Step ES13) a sequence of instructions similar to the following:

- Operation Code for "Select Component"
- Component ID (i.e. the ID of the applet panel)
- Operation Code for "Store Byte Array"
- Length of Byte Array
- Byte Array Data (containing the paint Instructions for the applet panel)
- Operation Code for "Assign Paint Instructions to Selected Components"

Referring now to Figure 9c, as the component manager 18 on the client computer 9 receives the above instructions, it locates a pointer to the "selected component" using the Component ID and the client component table 22; creates a new byte array containing the received byte array data; stores the byte array in the selected component as paint instructions (Step ES14); and calls the "repaint" method of the selected component (i.e. the applet panel client component 20b) (Step ES15). When the client windowing system 13 calls the "paint" method of the applet panel component 20b, the "paint" method calls the component painter 21, passing to it the graphics object received from the client windowing system 13 and the stored byte array of paint instructions (Step ES16). The component painter 21 decodes the byte array of paint instructions and appropriately calls methods of the graphics object in a sequence and manner identical to the original calls made by the "paint" method of the target application's applet panel running on the server computer 8.

Meanwhile, on the server computer 8, after transmission of the paint instructions (at Step ES13), the enabler program calls the "init" method of the applet panel component 3b (Step ES17). It is normal practice for the "init" method of an applet panel to exercise

application logic to deploy an initial set of windowing components to begin the user's "experience" of the application. For the present example, the application 1 will create the "Start" button 31 and the input and output text fields 29 and 30.

In a conventional manner, the application 1 creates a button component 3d (Step ES18), sets its attributes (for example its color, size, location, label, etc.), and calls the "add" method of the applet panel component 3b to add the new button component 3d onto the applet panel (Step ES19). By adding the button to the applet panel, the remote component toolkit 14 is called to create a remote component peer for the button (Step ES20). When that button remote component peer 17d is created, it gets the button component's attributes (Step ES21) and sends component creation instructions to the client computer 9 (Step ES22).

After the application 1 has created the "Start" button component 3d and added it to the applet panel component 3b, the application 1 proceeds to create and add the remaining components 3c and 3e (Step ES23) in a manner similar to the creation of the "Start" button component 3d as described above. When all of the necessary components 3c-3e have been added to the applet panel component 3b, the "init" method completes and returns control to the enabler program (Step ES24) which flushes the communications output buffers and passes control to the remote event dispatcher 15 which blocks waiting for a transmission to be received from the client computer 9.

Looking back to Figure 9b, notice that the enabler program starts a "Flush Timer" thread prior to creating the applet panel component 3b at ES9, that is, before any application code is executed on the server computer 8. The purpose of the flush timer thread is to ensure that the physical transmission of component instructions generated by remote component peers is not excessively delayed. The output stream of the communications software can be expected to buffer data to be transmitted. Physical transmission only occurs when the buffer is filled or a "flush" operation is performed on the output stream. In many applications, the communications software may effect physical transmission of buffer data without intervention of the flush timer thread. Such applications are those that initialize quickly, respond to events quickly, and do not start their own threads that manipulate windowing components. However, in applications that have long processing delays or launch threads that manipulate windowing components, the flush timer thread plays an important role by flushing the output stream in such a manner as to ensure that component instructions are not delayed any longer than a prescribed amount of time. Ideally a decision of when to flush the output stream should be

a function of the age of both the oldest and newest component instructions currently buffered in the output stream, and whether or not the remote event dispatcher 15 is reading from the communications input stream. By considering these factors, an acceptable trade off can be made between user perception of delays and physical network traffic.

Referring now to Figure 9d, as the user types the number "100" into the input field 29, each key pressed by the user (i.e. "1" ... "0" ..."0") results in associated events being generated and passed to the input TextField client component 20c (Step ES25). As none of these events are considered to be significant, the event filter 23 does not pass the events to the event transmitter 25. Instead, the keys pressed by the user are accumulated by and within the input TextField component peer 6c. The event filter 23 will, however, add a reference to the input TextField client component 20c to a "changed components" list.

Subsequently, the user moves a pointer (for example, a mouse) to another component (in this case, it will be assumed to be the "Start" button 31) and depresses a key of the pointer to "click" on that component. This action produces a "button click" event that is passed to the button client component 20d and forwarded to the event filter 23 (Step ES26). As a first step, the "button click" event is analyzed by the event filter 23 to determine whether it is significant. Upon determination that the event is significant, the event filter 23 calls the change transmitter 24 (Step ES27) to send component changes if needed. The change transmitter 24 checks the "changed components" list and determines that the input TextField 29 has been changed, and therefore must be updated on the server computer 8. Accordingly, the change transmitter 24 gets the input TextField's component ID from the client component table 22, reads the text data from the input TextField client component 20c, and transmits an "activity packet" (Step ES28) with contents similar to the following:

- Length of the entire Activity Packet
- Component ID of the input TextField client component 20c
- Operation Code for "Text Change"
- Length of following Text Data (i.e. 3)
- Text Data now contained in the input TextField client component 20c (i.e. "100")

When this information is received, the remote event dispatcher 15 extracts the component ID from the activity packet (Step ES30), locates the corresponding remote

component peer in the component table 16 (in this case the input TextField remote component peer 17c for the input TextField), and passes the activity packet data to the input TextField remote component peer 17c for processing (Step ES31). The input TextField remote component peer 17c, after examination of the operation code, extracts the text data (in this case the string "100"), and stores it within the peer for later retrieval by the application 1 (Step ES32).

Once transmission of the contents of the input field client component 20c is complete, the Change transmitter 24 returns control to the event filter 23, which calls the event transmitter 25, passing to it the "button click" event for transmission (Step ES33). The event transmitter 25 constructs and transmits an event-type activity packet (Step ES34) with contents similar to the following:

- Length of the entire Activity Packet
- Component ID of the Button component
- Operation Code for "Event"
- Event Type Code for "Button Click"

Referring now to Figure 9e, when the "button click" activity packet is received, the remote event dispatcher 15 extracts the component ID from the activity packet (Step ES35), locates the button remote component peer 17d in the component table 16, and passes the activity packet data to the button remote component peer 17d for processing (Step ES36). The button remote component peer 17d, after examination of the operation code and event type code, creates an event object representing the "button click" event and calls the event-processing method of its associated button component 3d, passing the event object as a parameter (Step ES37).

As a result of receiving the button click event object, the button component's event processing method invokes application logic which, in accordance with the present example, (a) reads the contents of the input TextField component 3c to determine the number of calculations to perform (Step ES38); (b) sets the button component's label to "Stop" (Step ES39); (c) starts an Application Thread to perform the repetitive calculations (Step ES40); and (d) returns control back to the remote event dispatcher 15 (Step ES41).

When the application 1 reads the contents of the input TextField component 3c, it does so by calling a "get text" method of the input TextField component 3c, which, in turn, calls a "get text" method of its corresponding input TextField remote component peer 17c.

The input TextField remote component peer 17c returns the string "100" which it had previously stored (at Step ES38).

When the application 1 sets the button's label to "Stop" (at Step ES39) by calling a "set label" method of the button component 3d, the button component 3d calls a "set label" method of its corresponding button remote component peer 17d. The "set label" method of the button remote component peer 17d transmits to the client computer 9 (Step ES42) a sequence of component instructions similar to the following:

- Operation Code for "Store text"
- Length of following Text
- Text (i.e. "Stop")
- Operation Code for "select component'
- Component ID (I.e. the ID of the button component)
- Operation Code for "set label of selected component using stored text"

When these instructions are received, the component manager 18 acts each of the instructions code in sequence by first storing the text, second looking up the component ID in the client component table 22 and storing the pointer to the selected component, and finally calling the "set labels" method of the selected component with the stored text (Step ES43) as a parameter. Thus in the present example, the component manager 18 obtains the component ID of the button client component 20d from the client component table 22, and then calls the "set labels" method of the button client component 20d with the stored text as a parameter. The "set labels" method of the button client component 20d calls the "set labels" method of the corresponding button component peer 6d in a conventional manner to effect rendering of the button 31 with the new label text on the monitor 28 of the client computer 9.

When the application 1 returns control to the remote event dispatcher 15 (at Step ES41), the remote event dispatcher 15 flushes the communications output buffers, ensuring that all instructions transmitted by the remote component peers are now physically sent. In the present example, this most likely results in the entire physical transmission of the instructions needed to set the button label to "stop".

Now, referring to Figure 9f, the application thread that was started by the application 1 as a result of the "button click" event (at Step ES40) begins operation by requesting the repainting of the applet panel for a new rendition of the progress bar 32.

The application thread does this by calling the "repaint" method of the applet panel component 3b (Step ES44), which in turn calls the "repaint" method of the corresponding applet panel remote component peer 17b. The applet panel remote component peer 17b proceeds to call the applet panel component's "paint" method (Step ES45) and send new paint instructions to the client computer 9 (Step ES46) in the same manner as described earlier when the applet panel remote component peer 17b was first created. Notice, however, that under the present circumstances, with component instructions being generated as a result of a thread of execution other than that of the remote event dispatcher 15 (i.e. the application thread), that physical transmission of data is guaranteed only as a result of the flushing of the output stream by the flush timer thread.

Following its initial repaint of the applet panel, the application thread proceeds to perform 100 iterations of its designated computational task, during which it periodically repaints the applet panel to show its progress through the iterations (Step ES47). When the iterations are complete, the application thread computes the elapsed time and calls the "Set text" method of the output TextField component 3e passing the elapsed time as a parameter (Step ES48). The output TextField component 3e then passes the elapsed time string to the "set text" method of its associated output TextField remote component peer 17d, which transmits (Step ES49) a series of component instructions similar to:

- Operation Code for "store text"
- Length of following Text
- Text (i.e. the elapsed time)
- Operation Code for "Select component'
- Component ID (I.e. the ID of the input text field component)
- Operation Code for "set text of selected component using stored text"

As described above, when these instructions are received, the component manager 18 acts each of the instructions code in sequence by first storing the text, second looking up the component ID in the client component table 22 and storing the pointer to the selected component, and finally calling the "set text" method of the selected component with the stored text as a parameter. Thus in the present example, the component manager 18 obtains the component ID of the output TextField client component 20e from the client component table 22, and then calls the "set text" method of the output TextField client component 20e with the stored text as a parameter. The "set text" method of the output TextField client component 20e calls the "set text" method of the

corresponding output TextField component peer 6e in a conventional manner to effect rendering of the output text field 30 with the new label text on the monitor 28 of the client computer 9 (Step ES50).

The application thread then sets the button label back to "Start" (Step ES51 and ES52) in a manner directly analogous to that described above for setting the button label to "Stop", and finally terminates its execution (Step ES53). Notice again, that since the remote windowing system 11 is unaware of the completion of the application threads activities, the flushing of the communications output stream is performed by the flush timer thread.

INDUSTRIAL APPLICABILITY

The present invention is applicable to the field of communications through computer networks, and in particular to the field of communications through the Internet. The present invention will be of particular utility for electronic commerce using such networks, and for so-called "network computers" which are designed to rely on server-based applications, rather than applications stored on the computer's own hard-disk, to provide users with functional programs.

CLAIMS:

1. In a network comprising at least one server computer and at least one client computer connected via a communications media, the client computer having a conventional windowing system capable of rendering components of an application on a monitor of the client computer, and further capable of passing user-initiated events to components of an application running on the client computer, a method of manipulating components of an application running on the server computer, the method comprising the steps of:

(a) monitoring the windowing system of the client computer, and, upon receipt of a user initiated event:

[i] encoding data indicative of the user initiated event to create an activity packet, the amount of data of the activity packet typically being smaller than that of the data indicative of the event;

[ii] transmitting the activity packet to the server computer;

[iii] receiving the activity packet in the server computer;

[iv] decoding the activity packet to obtain event data indicative of the user initiated event;

[v] passing the event data to the application running on the server computer to process the event;

(b) monitoring the application running on the server computer, and, upon receipt of component change data indicative of changes in a component:

[i] transmitting the component change data to the client computer;

[ii] receiving the component change data in the client computer;

[iii] passing the component change data to components running on the client computer for rendering on the monitor of the client computer.

2. The method of claim 1, wherein steps (a) and (b) of claim 1 are conducted in sequence as an event/response cycle initiated by receipt of a user initiated event at step (a) of claim 1.

3. The method of claim 1, wherein step (a) of claim 1 includes the step of filtering the user-initiated event to determine whether or not the user-initiated event is significant, steps a[i] through b[iii] of claim 1 being executed only when it is determined that the user-initiated event is in fact significant.

4. The method of claim 1, wherein step (a) of claim 1 further includes the step of storing user-induced component changes, and, prior to step a[i] of claim 1:

(1) checking components for user-induced changes;

(2) when user induced changes are found, encoding the user induced changes to create an activity packet concerning the user induced changes;

(3) transmitting the activity packet to the server computer;

(4) receiving the activity packet in the server computer;

(5) decoding the activity packet to obtain the user-induced changes; and

(6) storing the user-induced changes in association with a selected component of the application.

5. A method of manipulating components on a remote computer via a network, the method comprising the steps of:

(a) receiving component change data concerning changes in a component from an application running on a first computer;

(b) transmitting encoded data indicative of the component change data to the remote computer;

(c) receiving the encoded data in the remote computer;

(d) decoding the encoded data in the remote computer to obtain component change data concerning desired component changes;

(e) passing the component change data to a selected component running on the remote computer to render the desired component changes.

6. The method of claim 5, wherein the encoded data comprises:

(a) a component ID indicative of a component to be changed, the component ID be used in the remote computer to select a desired component to be changed;

(b) an Op Code indicative of the type of change to be effected, the Op-Code being used in the remote computer to trigger a method of the desired component; and

(c) data serving as parameters for the method of the desired component triggered in response to the Op-Code.

7. A method of manipulating components on a remote computer comprising the steps of:

(a) receiving a user initiated event from a component running on a first computer;

(b) encoding the event to create an activity packet;

(c) transmitting the activity packet to a second computer;

(d) receiving the activity packet in the second computer;

(e) decoding the activity packet to obtain the event;

(f) passing the event to a selected component of an application running on the second computer to process the event.

8. The method of claim 7, wherein the activity packet comprises:

(a) a component ID indicative of a component affected by the user-initiated event; and

(b) an Op Code indicative of the user-initiated event.

9. In a network comprising at least one server computer and at least one client computer connected via a communications media, a method of starting an application for use by a user of a client computer, the method comprising the steps of:

(a) preliminarily storing the application, a proxy application and an enabler program on a server computer;

(b) running the enabler program on the server computer to continuously monitor the communication media for a network connection attempt initiated from a client computer;

(c) transmitting the proxy application from the server computer to a client computer via the network and initializing the proxy application on the client computer;

(d) establishing, by means of the proxy application, a network connection with the enabler program on the server computer;

(e) initializing the application on the server computer in response to establishment of a network connection with the enabler program;

(f) creating, on the server computer, one or more components in accordance with start-up logic of the application;

(g) creating a respective remote component peer for each component;

(h) transmitting, by means of each remote component peer, component creation instructions to the proxy application running on the client computer;

(i) creating, on the client computer, a respective client component corresponding to each component created on the server computer, on the basis of the component creation instructions; and

(j) rendering on a monitor of the client computer, client components as required by the intended initial condition of the application so as to begin a user's experience of the application.

10. The method of claim 9, wherein the step of initializing the proxy application includes detecting characteristics of the client computer, and transmitting to the server computer data indicative of the detected characteristics.

11. The method of claim 9, wherein the step of creating, on the server computer, one or more components, includes, for each paintable component, defining and storing a respective byte array of encoded paint instructions corresponding to the paintable component.

12. The method of claim 11, wherein the step of transmitting component creation instructions to the proxy application includes, for each paintable component, the steps of:

transmitting the byte array of encoded paint instructions to the client computer; receiving the byte array in the client computer; and storing the byte array in the client computer in association with a corresponding client component.

13. The method of claim12, wherein the step of rendering client components on a monitor of the client computer comprises painting each paintable component in accordance with its respective byte array of encoded paint instructions.

14. A method of manipulating components through a network, comprising:

(a) receiving an event from a windowing system of a client computer, the event being initiated by a user of the client computer;

(b) encoding event data indicative of the user initiated event to create an activity packet;

(c) transmitting the activity packet to a server computer through the network;

(d) receiving the activity packet in the server computer;

(e) decoding the activity packet to obtain the event data indicative of the user initiated event;

(f) calling one or more methods of a component on the server computer in accordance with the event data;

(g) Receiving component change data indicative of changes in a component running on the server computer;

(h) transmitting the component change data to the client computer;

(i) receiving the component change data in the client computer; and

(j) calling one or more methods of a component running on the client computer in accordance with the component change data.

15. The method of claim 14, wherein a flush timer thread running on the server computer is used to force timely transmission of the component change data to the client computer.

16. The method of claim 14, wherein the activity packet containing encoded data indicative of a user-initiated event comprises:

(a) a component ID indicative of a component affected by the user-initiated event;

(b) an Operation Code indicating that the activity packet contains information of a user-initiated event; and

(c) an event-type code indicative of the type event initiated by the user.

17. The method of claim 14, wherein the step of transmitting component change data to the client computer comprises the steps of :

(a) transmitting an operation code indicative of the component change;

(b) if required, transmitting any data associated with the component change;

(c) transmitting an operation code for "select component'

(d) transmitting a Component ID indicative of a specific component affected by the component change; and

(e) transmitting an operation code indicative of a functional operation to be performed on the component identified by the component ID.

18. The method of claim 17, wherein the step of receiving the component change data in the client computer, comprises sequentially receiving and processing each transmission described in claim 4.

19. The method of claim 14, wherein the step of receiving a user initiated event includes determining whether or not the user-initiated event is significant; steps b-j of claim 14 being executed only when the user-initiated event is determined to be significant.

20. The method of claim 19, further comprising the step of, when the user initiated event is determined to be insignificant, storing data indicative of induced component changes related to the insignificant event.

21. The method of claim 19, further comprising, when a user initiated event is determined to be significant, the steps of:

(a) determining whether or not data indicative of induced component changes related to prior insignificant events have be previously stored, and, when it is determined that such data have been stored;

(b) encoding the data indicative of induced component changes to create an activity packet;

(c) transmitting the activity packet to the server computer through the network;

(d) receiving the activity packet in the server computer;

(e) decoding the activity packet to obtain the data indicative of the induced component changes; and

(f) storing the data indicative of the induced component changes in association with a component running on the server computer.

22. The method of claim 21, wherein the encoded data indicative of induced component changes is transmitted to the server computer prior to transmitting the event data indicative of the user initiated event.

23. The method of claim 21, wherein the activity packet containing encoded data indicative of induced component changes comprises:

(a) a component ID indicative of a component affected by the induced component change;

(b) an Operation Code indicative of a type of the induced component change; and

(c) if required, any data stored in association with the affected component as a result of the induced component change

24. In a network comprising a server computer and a client computer connected by a network media allowing two-way transmission of data between the server and client computers, and wherein each computer includes a conventional windowing system capable of facilitating interaction between components of an application running on each computer and a respective user of each computer, a method for manipulating components of an application running on the server computer in response to inputs provided by a user of the client computer, with minimum network traffic, the method comprising:

(a) providing a proxy application on the client computer capable of emulating, on the basis of instructions codes received from the server computer, components of the application running on the server computer, the

proxy application being further capable of receiving data and user-initiated events from the windowing system of the client computer, and transmitting an activity packet indicative of the data and user-initiated events input to the server computer;

(b) providing a remote windowing system on the server computer capable emulating, on the basis of codes received from the client computer, data and user-initiated events received from the windowing system of the client computer, the remote windowing system being further capable of receiving data indicative of changes in components of the application on the server computer, and transmitting instruction codes indicative of such component changes to the client computer;

(c) whereby components of an application running on the server computer can be manipulated on the basis of data and user-initiated events provided by a user of the client computer, and network traffic is limited to transmission of activity packets indicative of such data and user-initiated events from the client computer to the server computer, and component change data indicative of component changes from the server computer to the client computer.

25. In a network comprising at least one server computer and a client computer connected for communication via a network media, a method of initializing an application on a server computer for use by a user of the client computer, the method comprising the steps of:

(a) providing and continuously running an enabler program on a server computer, the enabler program being capable of monitoring the network for, and accepting, network connection requests from the client computer;

(b) retrieving an object class definition of a proxy application into the client computer, and initializing the proxy application in the client computer;

(c) using the proxy application to establish a network connection with the enabler program running on the server computer;

(d) when the enabler program on the server to accepts a network connection from the proxy application, using the enabler program to create and initializing an instance of the application's startup logic on the server;

(e) using the application's startup logic to create components consistent with the application's intended form of operation;

(f) as each component is created, creating a respective remote component peer on the server computer;

(g) using the remote component peer to transmit component creation instructions to the proxy application on the client computer; and

(h) using the component manager to create a client component on the client computer on the basis of the component creation instructions received from the server computer, the client component corresponding to and emulating a respective component on the server computer.

26. The method of claim 25, wherein the network is a TCP/IP network, and the client computer communicates with the server computer using a browser program.

27. The method of claim 26, wherein a user of the client computer triggers initialization of the application by clicking on a link which causes the browser to retrieve, from an HTTP server, an HTML page containing an HTML applet tag which identifies the object class definition of the proxy application; the browser being responsive to the applet tag to retrieve the object class definition of the proxy application from an HTTP server and create and initialize an instance of the proxy application object in a conventional manner.

28. The method of claim 26, further comprising, when the proxy application establishes a network connection with the server computer, the step of using the proxy application to detect and transmits data indicative of functional characteristics of the client computer.

29. The method of claim 28, wherein the data indicative of functional characteristics of the client computer includes the type and version of the operating system and the resolution of a monitor screen of the client computer.

30. The method of claim 28, further comprising, when the enabler program initializes the application startup logic on the server computer, the step of using the enabler program to replace the conventional windowing toolkit of the server computer with a remote component toolkit that is compatible with the proxy application running on the client computer and the detected functional characteristics of the client computer, whereby

remote component peers compatible with the proxy application and functional characteristics of the client computer are created during subsequent creation of components of the application on the server computer.

31. The method of claim 27, wherein the enabler program and application, the HTML page, and the object class definition of the proxy application are each stored of respective different server computers on the network.

32. The method of claim 27, wherein any two or more of the enabler program and application, the HTML page, and the object class definition of the proxy application are stored on the same server computer.

33. In a network comprising a server computer and a client computer connected by a network media allowing two-way transmission of data between the server and client computers, and wherein each computer includes a conventional windowing system capable of facilitating interaction between components of an application running on each computer and a respective user of each computer, a system for manipulating components of an application running on the server computer in response to inputs provided by a user of the client computer, with minimum network traffic, the system comprising:

(a) a proxy application running on the client computer, the proxy application being capable of emulating, on the basis of codes received from the server computer, components of the application running on the server computer, the proxy application being further capable of receiving data and user-initiated events from the windowing system of the client computer, and transmitting an activity packet indicative of data and user-initiated events to the server computer;

(b) a remote windowing system running on the server computer, the remote windowing system being capable emulating, on the basis of codes received from the client computer, data and user-initiated events received from the windowing system of the client computer, the remote windowing system being further capable of receiving data indicative of changes in components of the application on the server computer, and transmitting instructions codes indicative of such component changes to the client computer.

34. A system as claimed in claim 33, wherein the proxy application comprises:

(a) at least one client component capable of interacting with the windowing system of the client computer in a conventional manner to facilitate rendering of each client component and reception of data and user-initiated events;

(b) a component manager capable of instantiating and manipulating each client component in response to instruction codes received from the server computer;

(c) an event handler responsive to each client component and capable of receiving event data indicative of a user initiated event received by a client component, and transmitting the event data to the server computer as an activity packet;

(d) a client component table containing component identifier and address information for each client component, the client component table providing a look-up table whereby the component manager can select a component on the basis of a component ID received from the server computer; and

(e) a component painter capable of interacting with each client component, and the windowing system on the client computer, to control the rendering of "paintable" components on a monitor of the client computer.

35. A system as claimed in claim 34, wherein the event handler comprises:

(a) an event filter responsive to each client component for determining whether or not a user-initiated event is significant;

(b) an event transmitter responsive to the event filter for encoding and transmitting to the server computer event data as an activity packet;

(c) a change transmitter responsive to the event filter for encoding and transmitting to the server computer induced component change data as an activity packet.

36. A system as claimed in claim 35, wherein the event transmitter is caused to transmit event data only when event filter determines that the user-initiated event is significant.

37. A system as claimed in claim 35, wherein the event filter stores data indicative of induced component changes when it a user-initiated event is determined to be insignificant.

38. A system as claimed in claim 37, wherein, when the event filter determines that a user-initiated event is significant, the event filter causes the change transmitter to encode and transmit induced component changes prior to causing the event transmitter to encode and transmit the event data.

39. A system as claimed in claim 33, wherein the remote windowing system comprises:

(a) a respective remote component peer corresponding to each component of an application running on the server computer, the remote component peer being capable of interacting with its corresponding component in a conventional manner to reflect changes in the component resulting from operation of the application and to pass user-initiated events to the component for processing, the remote component peer being responsive to component changes to transmit change codes indicative of the component changes to the client computer;

(b) a remote component toolkit comprising object classes for creating a remote component peer to correspond with a respective component, during creation of each component of the application;

(c) remote event dispatcher capable of receiving activity packets from the client computer, decoding each activity packet to obtain event data, and passing the event data to a selected remote component peer as an event; and

(d) a component table including component identifiers and addresses for each remote component peer, whereby the remote event dispatcher can readily select a remote component peer on the basis of the event data received from the client computer.

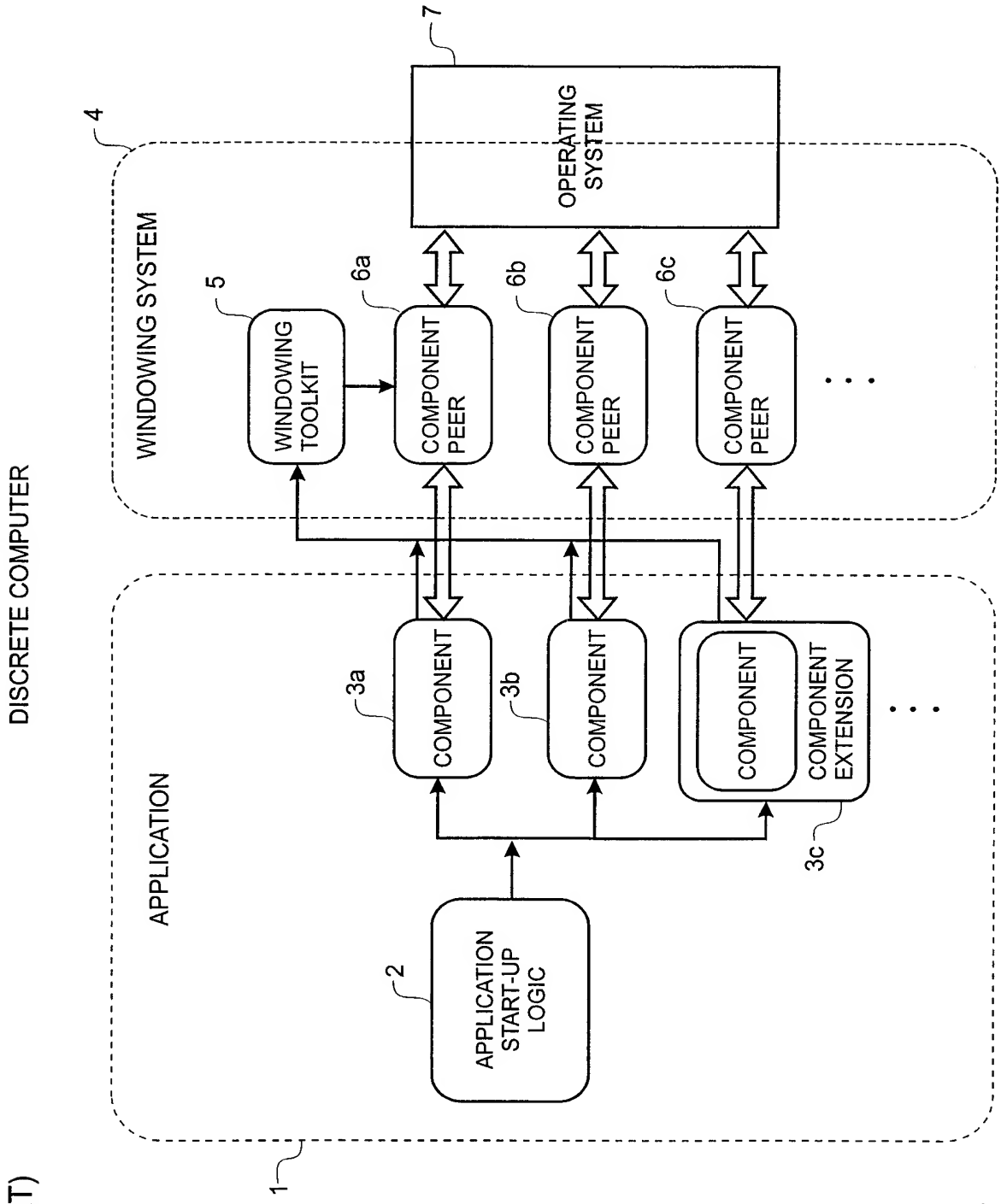


Figure 1
(PRIOR ART)

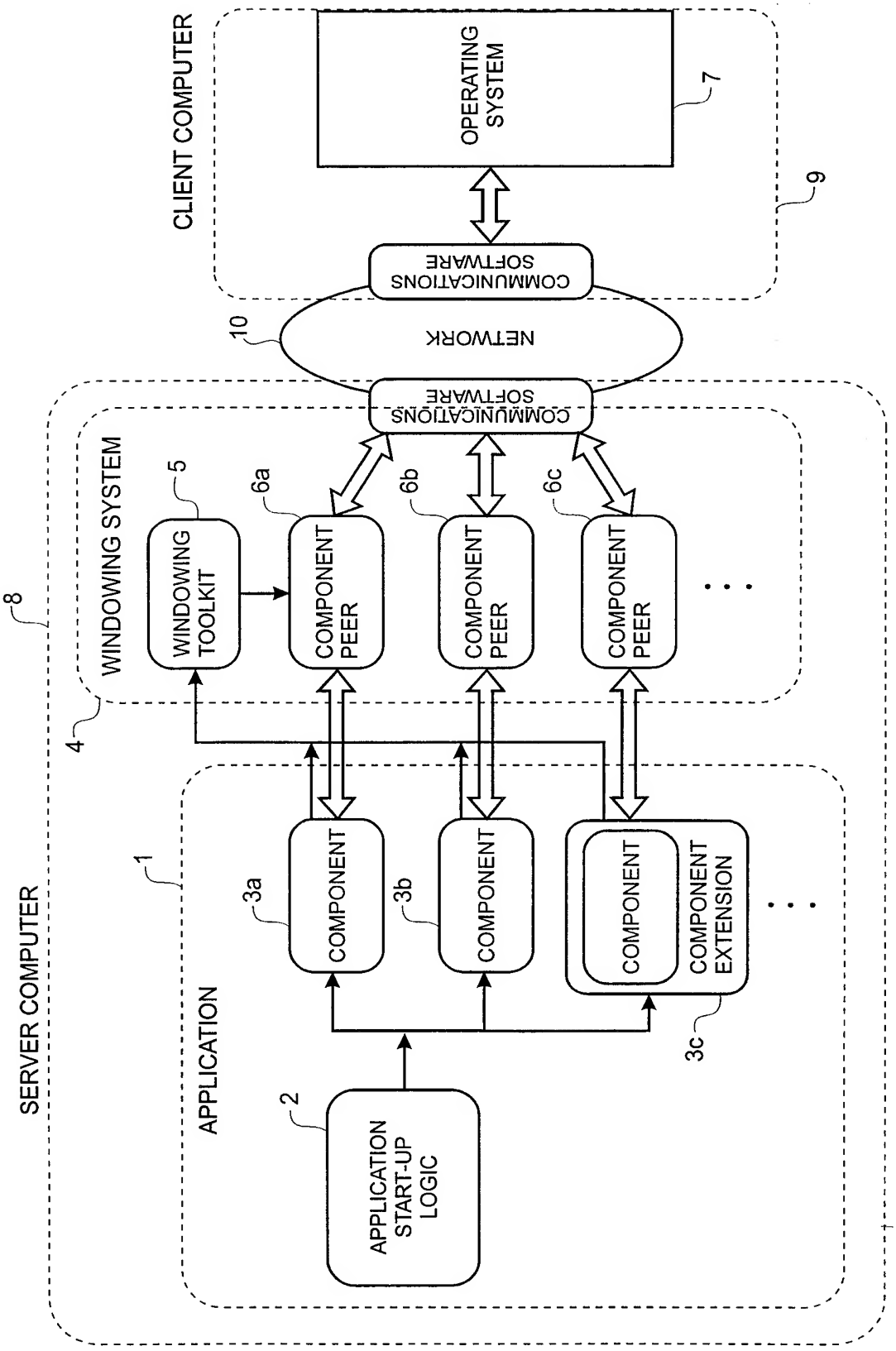


Figure 2
(PRIOR ART)

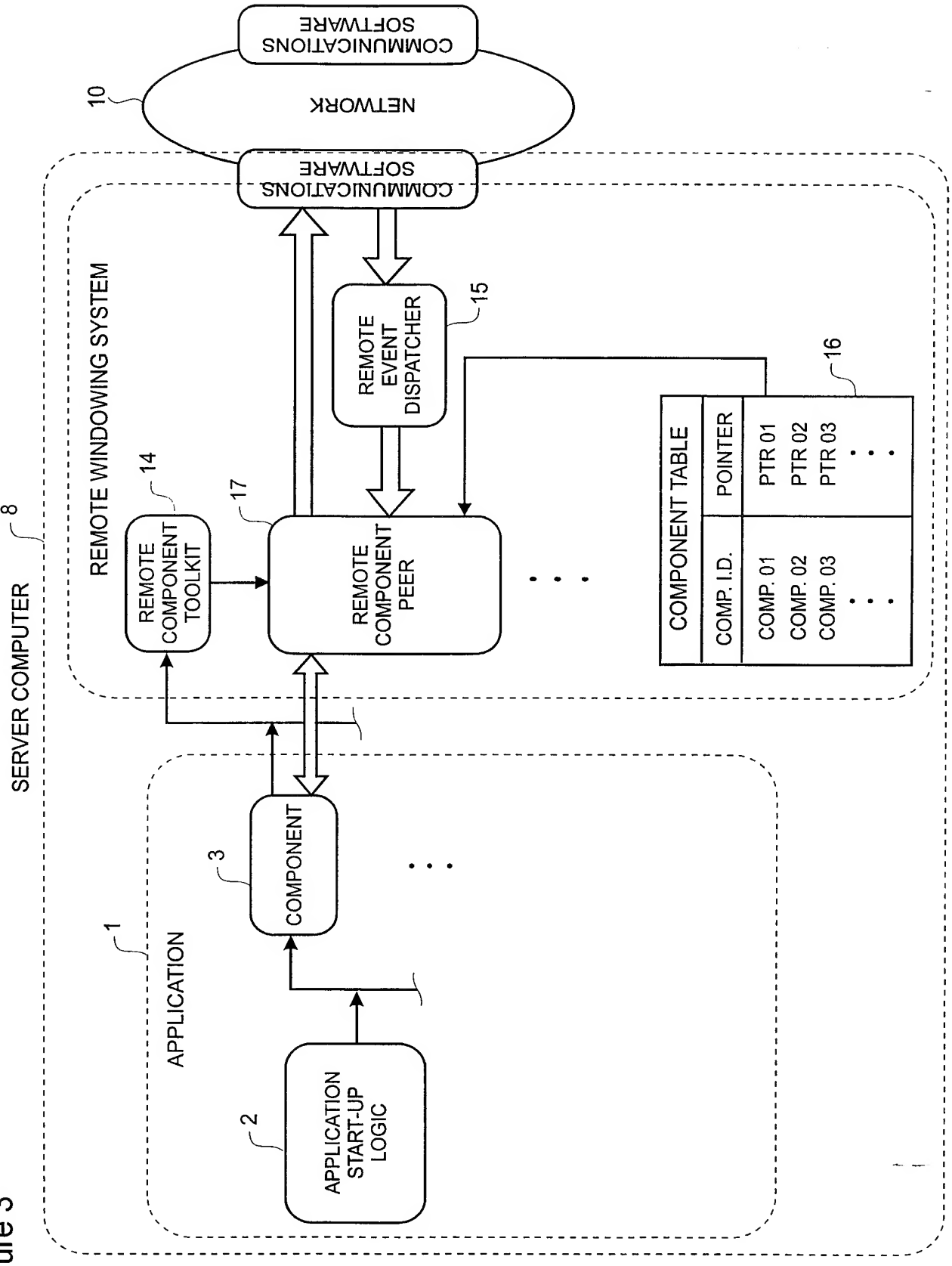

SERVER COMPUTER
8
10
NETWORK
COMMUNICATIONS SOFTWARE
COMMUNICATIONS SOFTWARE
REMOTE WINDOWING SYSTEM
REMOTE EVENT DISPATCHER
15
16
COMPONENT TABLE
COMP. I.D.
POINTER
COMP. 01
COMP. 02
COMP. 03
PTR 01
PTR 02
PTR 03
14
REMOTE COMPONENT TOOLKIT
17
REMOTE COMPONENT PEER
APPLICATION
1
3
COMPONENT
2
APPLICATION START-UP LOGIC

Figure 3

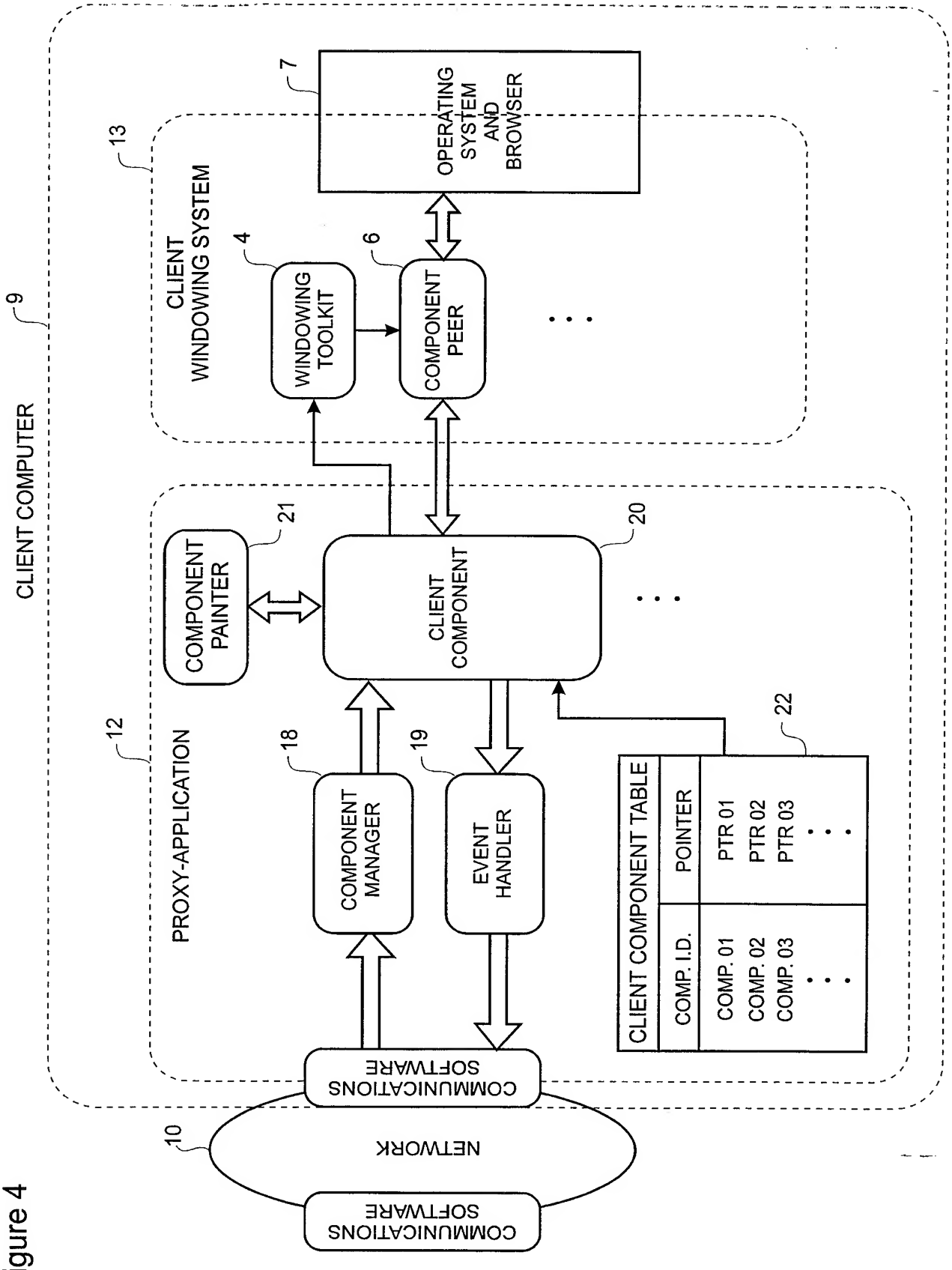

CLIENT COMPUTER
9
CLIENT WINDOWING SYSTEM
13
OPERATING SYSTEM AND BROWSER
7
WINDOWING TOOLKIT
4
COMPONENT PEER
6
PROXY-APPLICATION
12
COMPONENT PAINTER
21
CLIENT COMPONENT
20
COMPONENT MANAGER
18
EVENT HANDLER
19
CLIENT COMPONENT TABLE
COMP. I.D.
POINTER
COMP. 01
COMP. 02
COMP. 03
PTR 01
PTR 02
PTR 03
22
COMMUNICATIONS SOFTWARE
NETWORK
10
COMMUNICATIONS SOFTWARE

Figure 4

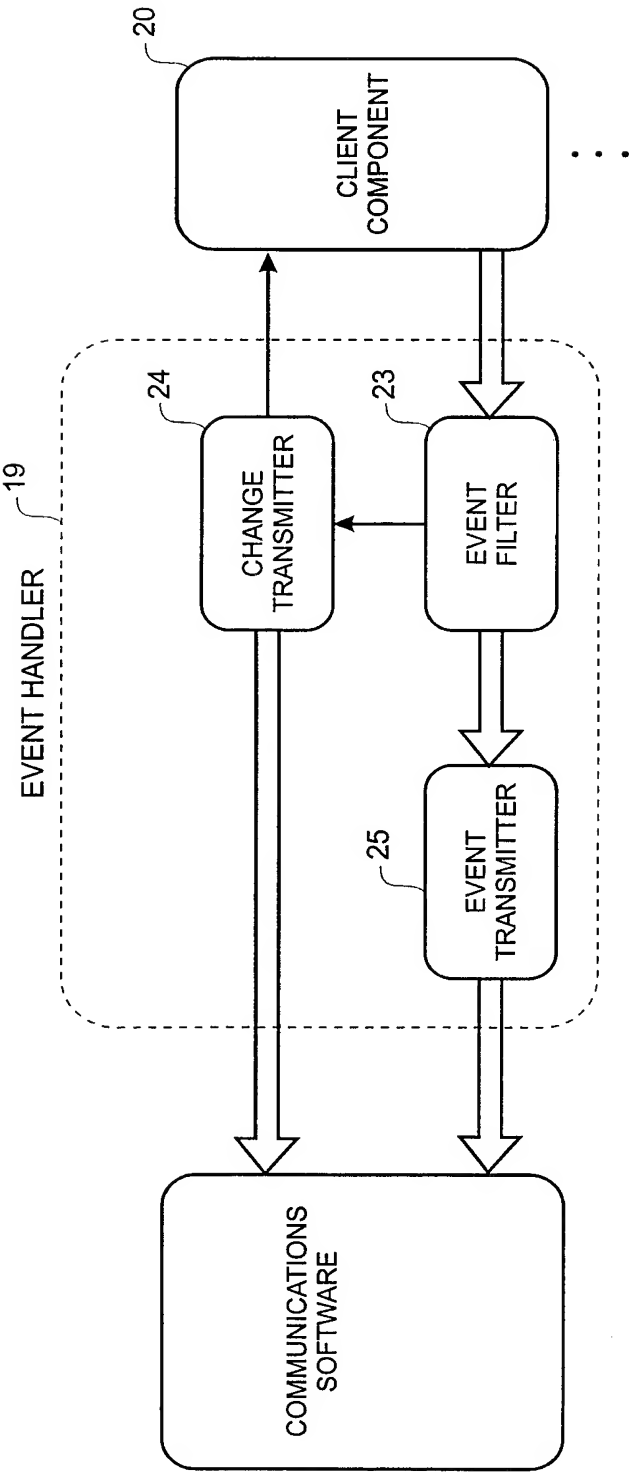

20
CLIENT COMPONENT
. . .
19
EVENT HANDLER
24
CHANGE TRANSMITTER
23
EVENT FILTER
25
EVENT TRANSMITTER
COMMUNICATIONS SOFTWARE

Figure 5

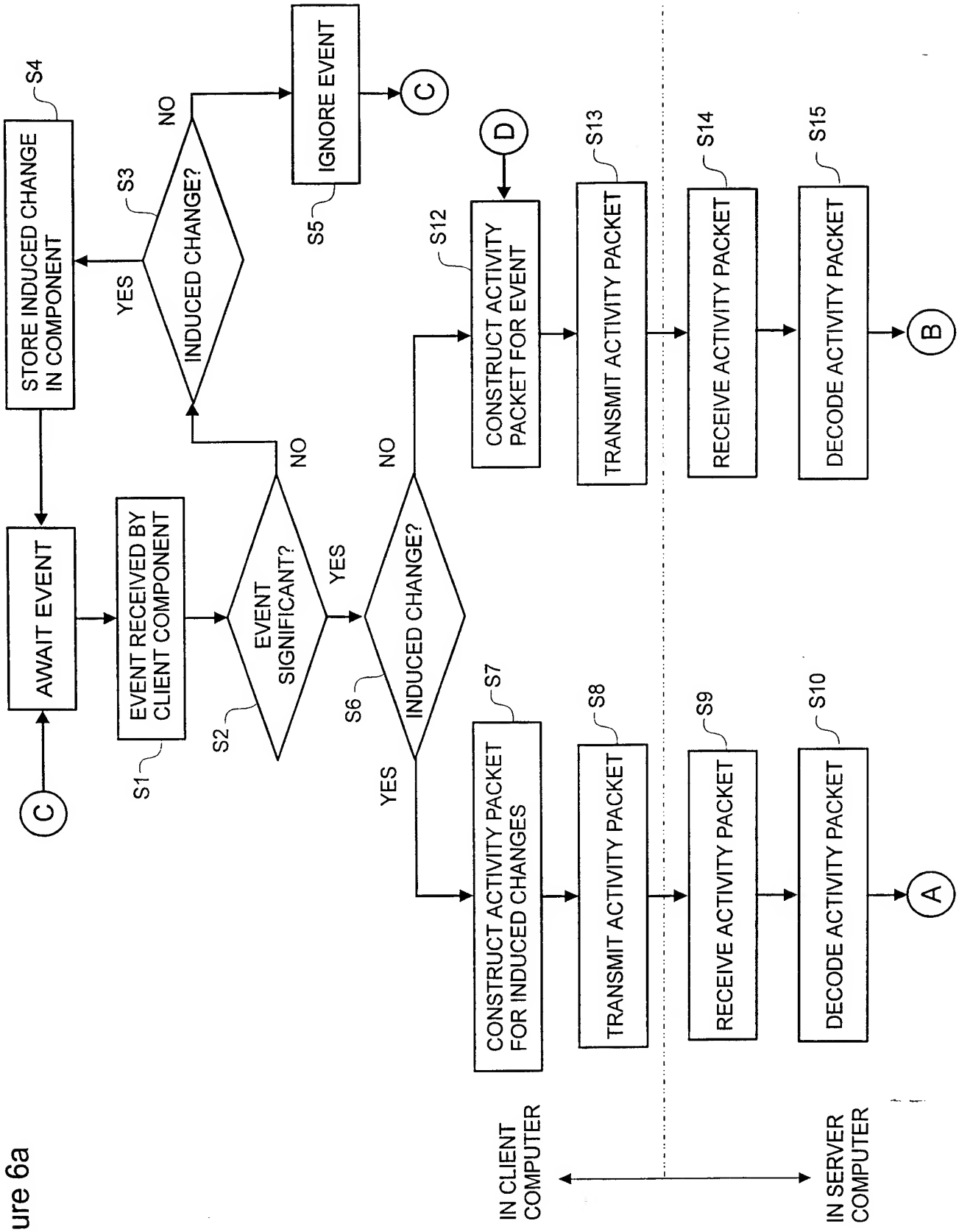

AWAIT EVENT
EVENT RECEIVED BY CLIENT COMPONENT
EVENT SIGNIFICANT?
INDUCED CHANGE?
CONSTRUCT ACTIVITY PACKET FOR INDUCED CHANGES
TRANSMIT ACTIVITY PACKET
RECEIVE ACTIVITY PACKET
DECODE ACTIVITY PACKET
CONSTRUCT ACTIVITY PACKET FOR EVENT
TRANSMIT ACTIVITY PACKET
RECEIVE ACTIVITY PACKET
DECODE ACTIVITY PACKET
INDUCED CHANGE?
STORE INDUCED CHANGE IN COMPONENT
IGNORE EVENT
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S12
S13
S14
S15
YES
NO
YES
NO
YES
NO
A
B
C
C
D
IN CLIENT COMPUTER
IN SERVER COMPUTER

Figure 6a

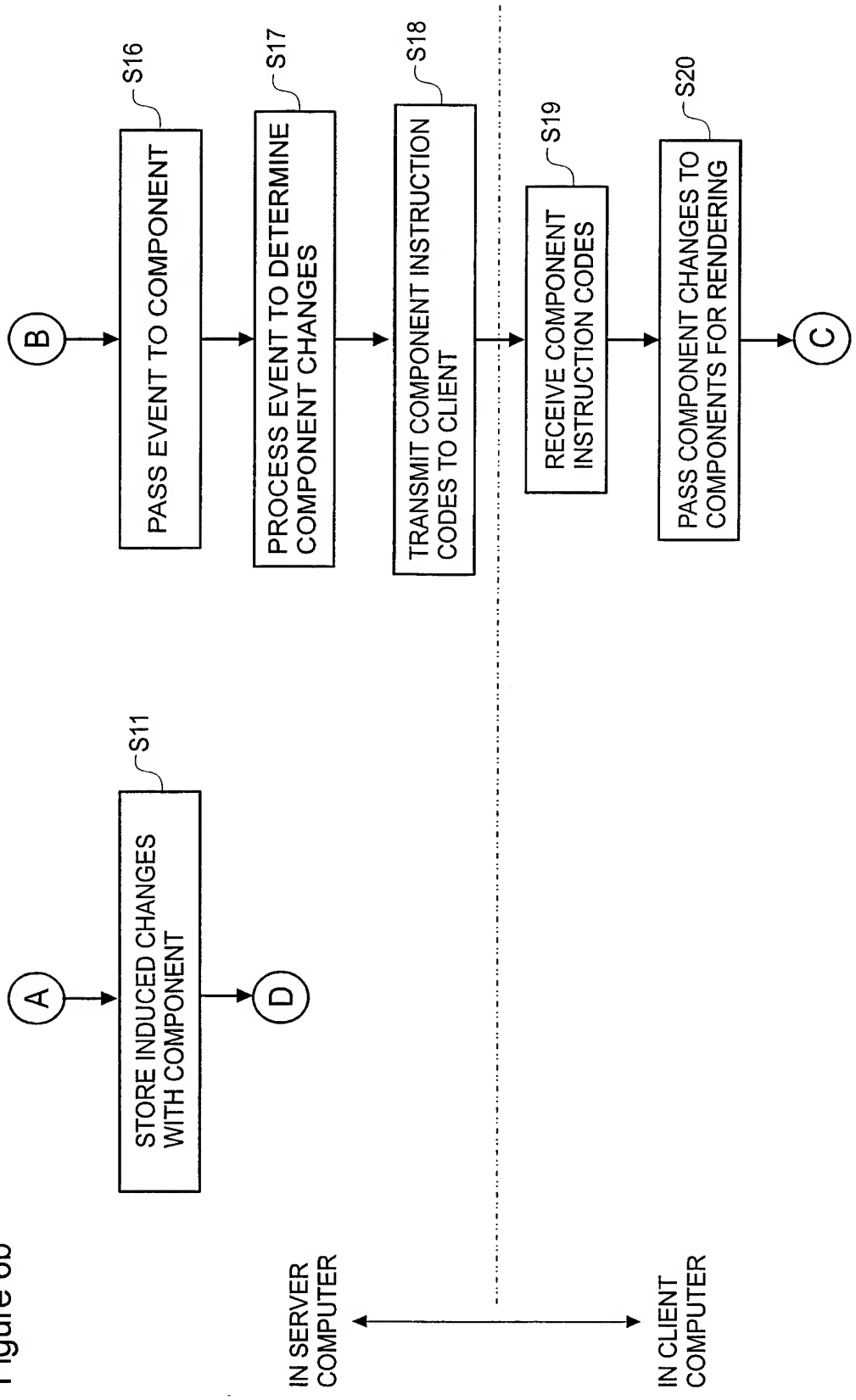

A
STORE INDUCED CHANGES WITH COMPONENT
S11
D
B
PASS EVENT TO COMPONENT
S16
PROCESS EVENT TO DETERMINE COMPONENT CHANGES
S17
TRANSMIT COMPONENT INSTRUCTION CODES TO CLIENT
S18
RECEIVE COMPONENT INSTRUCTION CODES
S19
PASS COMPONENT CHANGES TO COMPONENTS FOR RENDERING
S20
C
IN SERVER COMPUTER
IN CLIENT COMPUTER

Figure 6b

Figure 7a

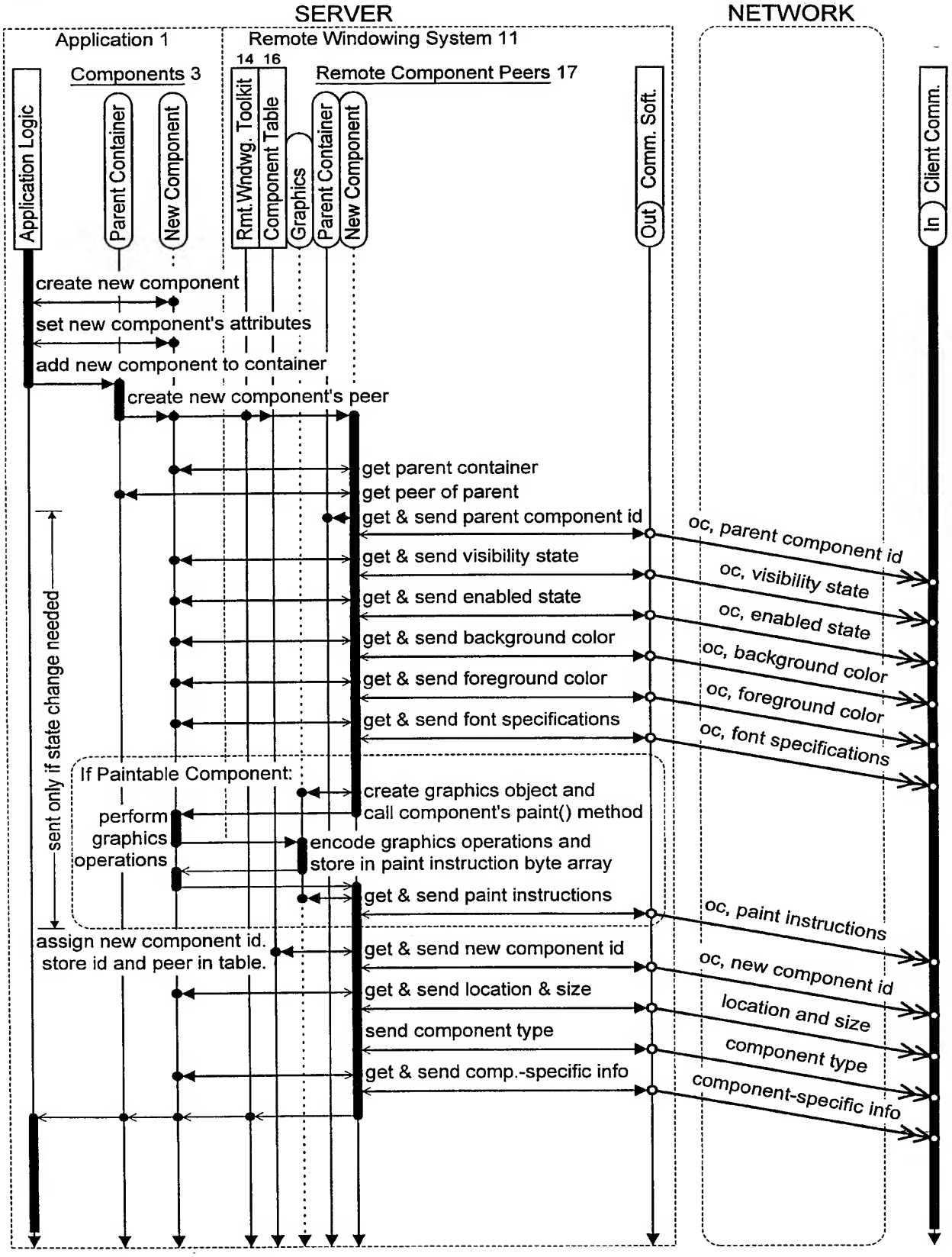
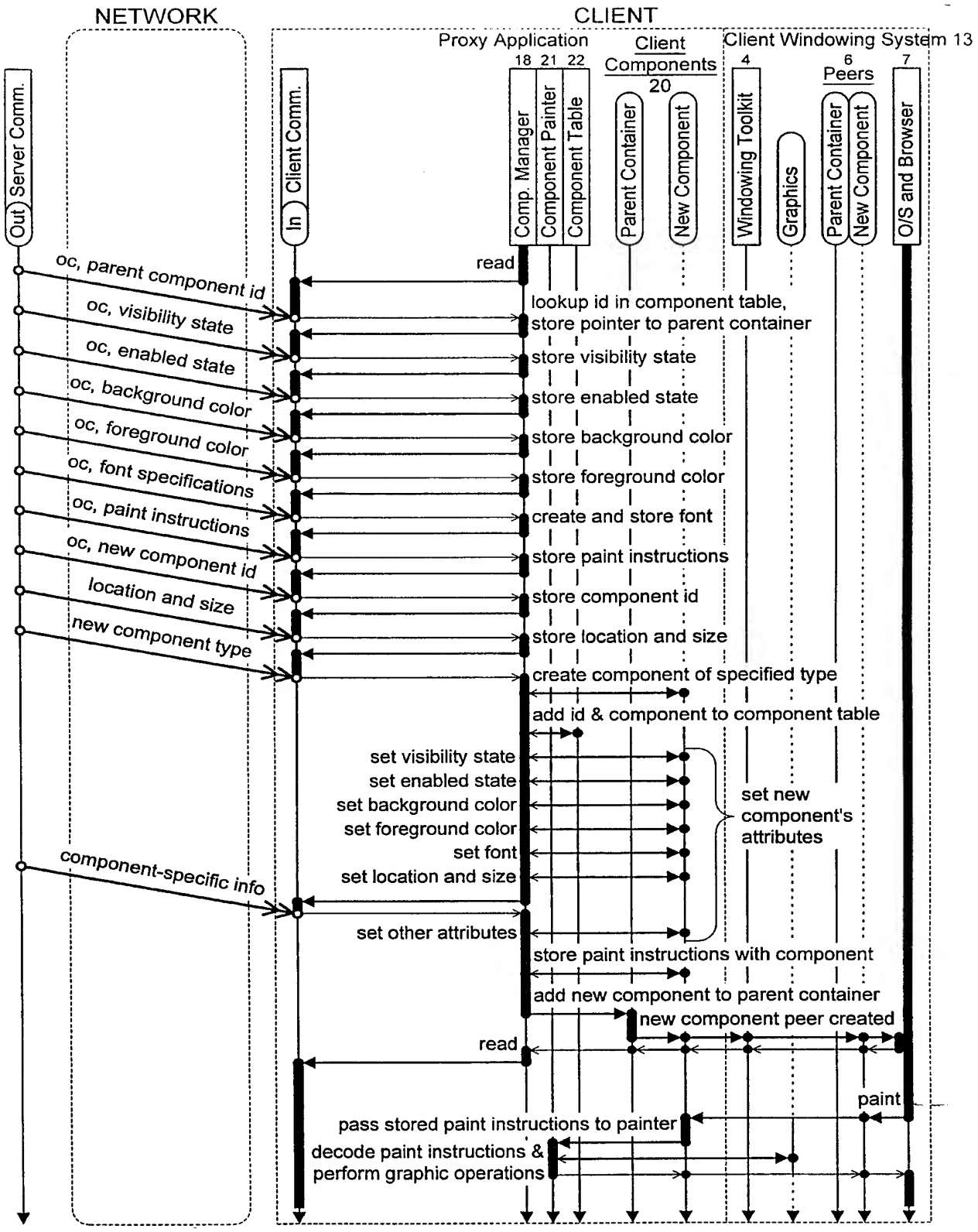

SERVER
NETWORK
Application 1
Remote Windowing System 11
Components 3
14 16
Remote Component Peers 17
Application Logic
Parent Container
New Component
Rmt.Wndwg. Toolkit
Component Table
Graphics
Parent Container
New Component
Out Comm. Soft.
In Client Comm.
create new component
set new component's attributes
add new component to container
create new component's peer
get parent container
get peer of parent
get & send parent component id
oc, parent component id
get & send visibility state
oc, visibility state
get & send enabled state
oc, enabled state
get & send background color
oc, background color
get & send foreground color
oc, foreground color
get & send font specifications
oc, font specifications
sent only if state change needed
If Paintable Component:
create graphics object and
call component's paint() method
perform graphics operations
encode graphics operations and
store in paint instruction byte array
get & send paint instructions
oc, paint instructions
assign new component id.
store id and peer in table.
get & send new component id
oc, new component id
get & send location & size
location and size
send component type
component type
get & send comp.-specific info
component-specific info

Figure 7b


NETWORK
CLIENT
Proxy Application
Client Components 20
Client Windowing System 13
Peers
18 21 22
4
6
7
Out Server Comm.
In Client Comm.
Comp. Manager
Component Painter
Component Table
Parent Container
New Component
Windowing Toolkit
Graphics
Parent Container
New Component
O/S and Browser
read
oc, parent component id
lookup id in component table,
oc, visibility state
store pointer to parent container
oc, enabled state
store visibility state
oc, background color
store enabled state
oc, foreground color
store background color
oc, font specifications
store foreground color
oc, paint instructions
create and store font
oc, new component id
store paint instructions
location and size
store component id
new component type
store location and size
create component of specified type
add id & component to component table
set visibility state
set enabled state
set background color
set foreground color
set font
set location and size
set new component's attributes
component-specific info
set other attributes
store paint instructions with component
add new component to parent container
new component peer created
read
paint
pass stored paint instructions to painter
decode paint instructions & perform graphic operations

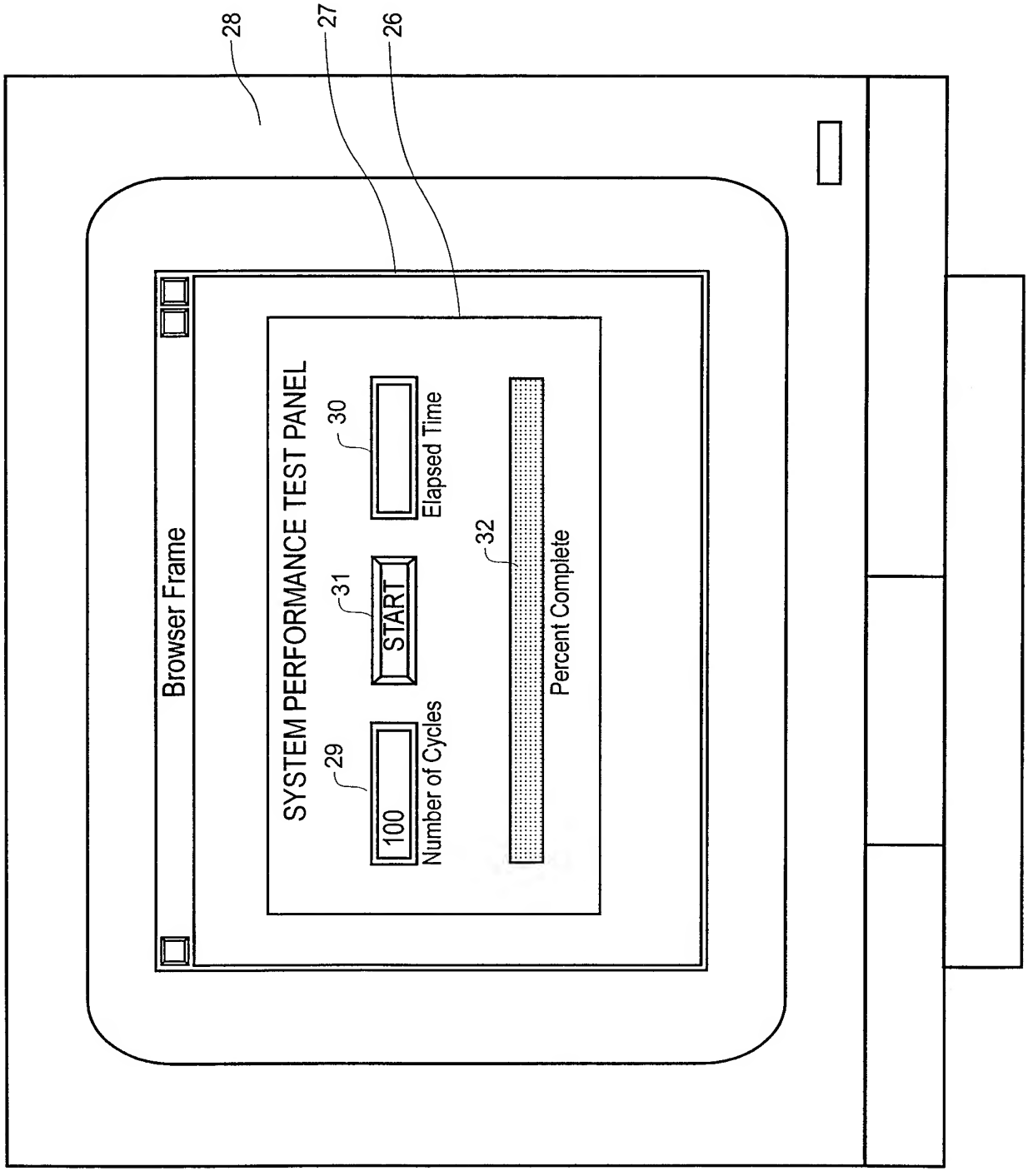

28
27
26
Browser Frame
SYSTEM PERFORMANCE TEST PANEL
30
Elapsed Time
31
START
32
Percent Complete
29
100
Number of Cycles

Figure 8

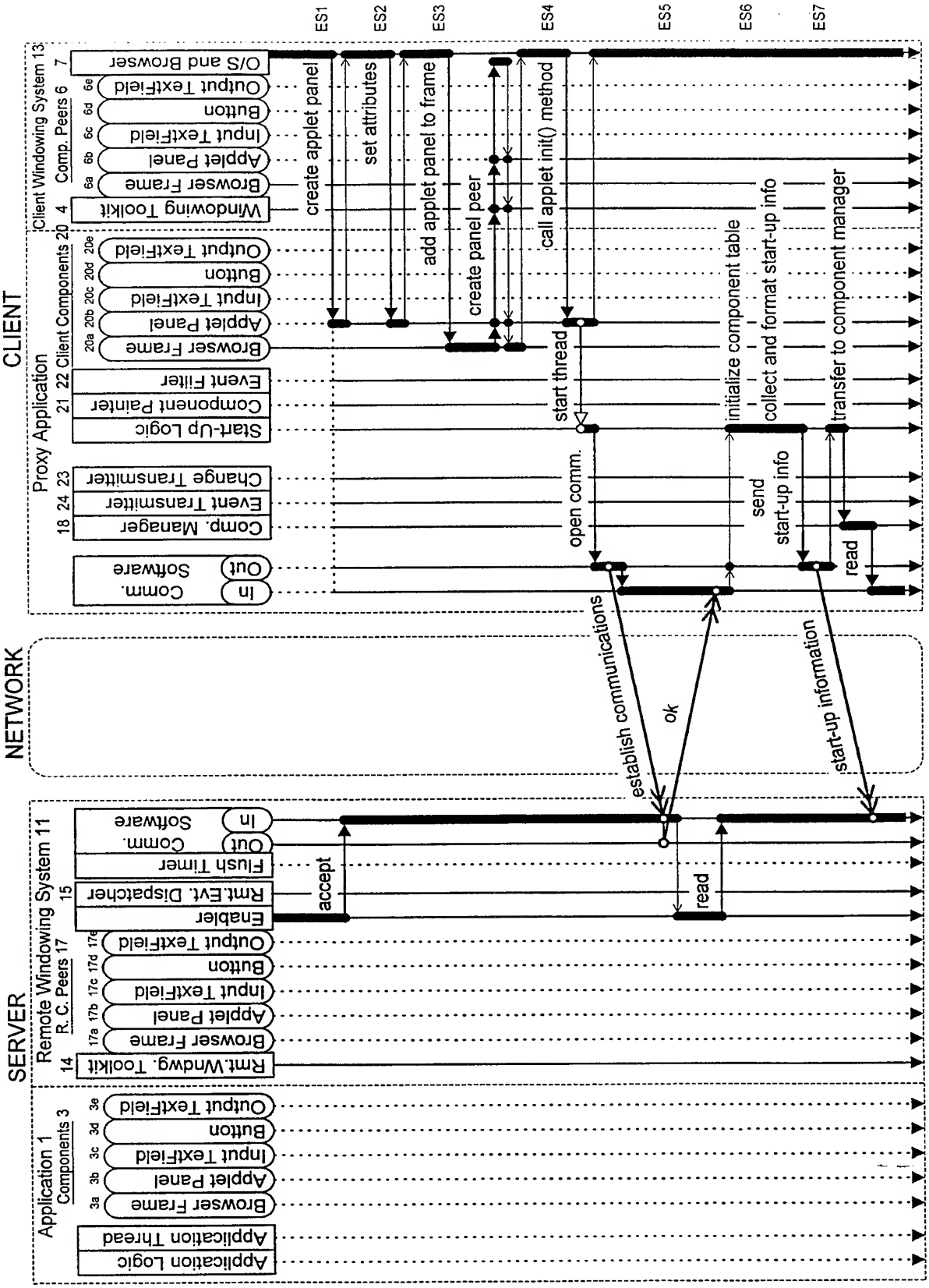

ES1
ES2
ES3
ES4
ES5
ES6
ES7
CLIENT
Client Windowing System 13
Comp. Peers 6
7 O/S and Browser
6e Output TextField
6d Button
6c Input TextField
6b Applet Panel
6a Browser Frame
4 Windowing Toolkit
Proxy Application
20 Client Components
20e Output TextField
20d Button
20c Input TextField
20b Applet Panel
20a Browser Frame
22 Event Filter
21 Component Painter
Start-Up Logic
23 Change Transmitter
24 Event Transmitter
18 Comp. Manager
Comm. Software
In
Out
create applet panel
set attributes
add applet panel to frame
create panel peer
call applet init() method
start thread
initialize component table
collect and format start-up info
transfer to component manager
open comm.
send start-up info
read
NETWORK
establish communications
ok
start-up information
SERVER
Remote Windowing System 11
R.C. Peers 17
14 Rmt.Wndwg. Toolkit
17a Browser Frame
17b Applet Panel
17c Input TextField
17d Button
17e Output TextField
15 Enabler
Rmt.Evt. Dispatcher
Flush Timer
Comm. Software
In
Out
accept
read
Application 1
Components 3
3a Browser Frame
3b Applet Panel
3c Input TextField
3d Button
3e Output TextField
Application Logic
Application Thread

Figure 9a

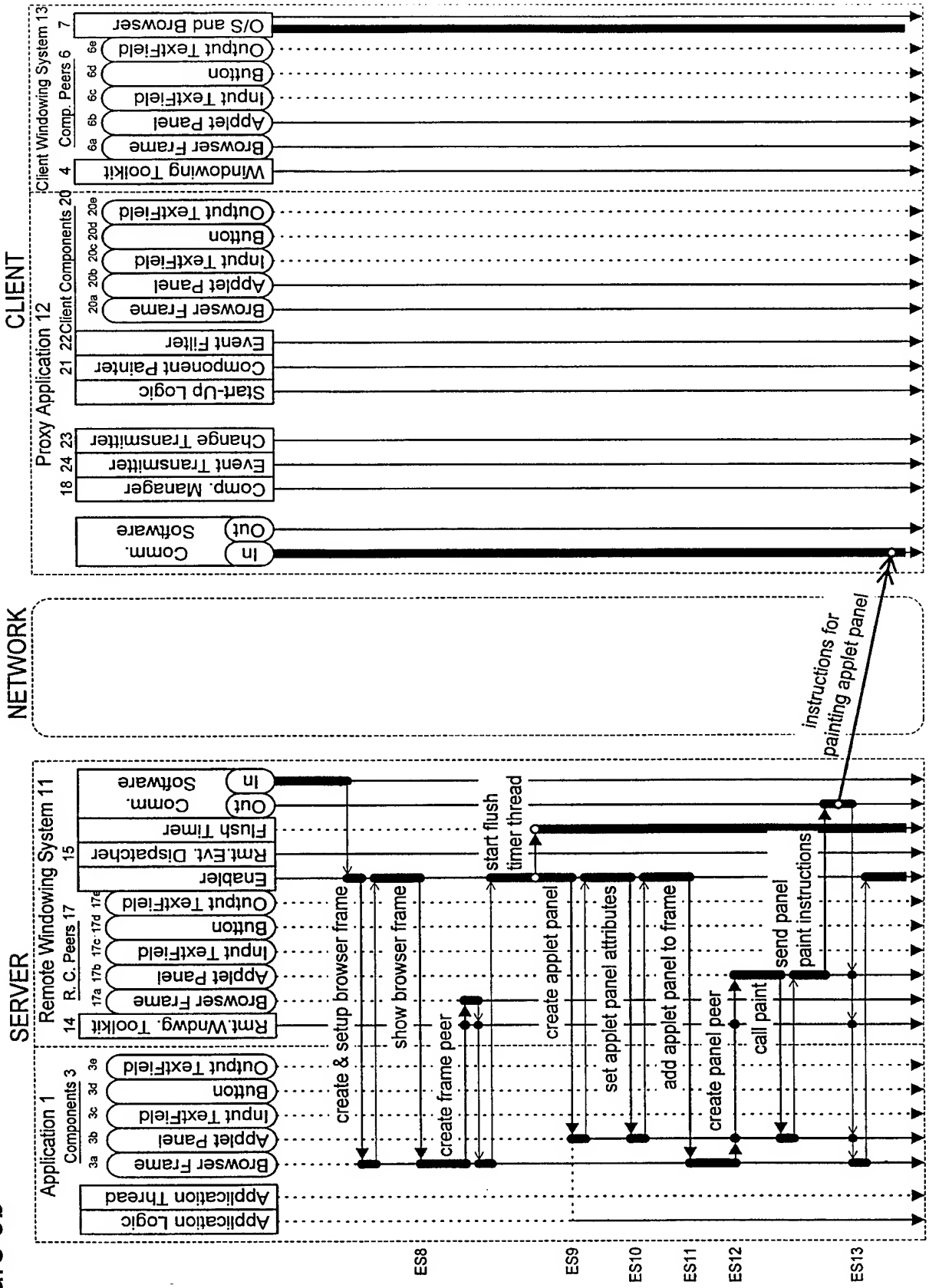

SERVER
CLIENT
NETWORK
Application 1
Components 3
3a Browser Frame
3b Applet Panel
3c Input TextField
3d Button
3e Output TextField
Application Logic
Application Thread
Remote Windowing System 11
14 Rmt.Wndwg. Toolkit
R.C. Peers 17
17a Browser Frame
17b Applet Panel
17c Input TextField
17d Button
17e Output TextField
15 Enabler
Rmt.Evt. Dispatcher
Flush Timer
Comm. Software
In
Out
Proxy Application 12
Client Components 20
20a Browser Frame
20b Applet Panel
20c Input TextField
20d Button
20e Output TextField
21 Component Painter
22 Event Filter
Start-Up Logic
23 Change Transmitter
24 Event Transmitter
18 Comp. Manager
Comm. Software
In
Out
Client Windowing System 13
4 Windowing Toolkit
Comp. Peers 6
6a Browser Frame
6b Applet Panel
6c Input TextField
6d Button
6e Output TextField
7 O/S and Browser
create & setup browser frame
show browser frame
create frame peer
start flush timer thread
create applet panel
set applet panel attributes
add applet panel to frame
create panel peer
call paint
send panel paint instructions
instructions for painting applet panel
ES8
ES9
ES10
ES11
ES12
ES13

Figure 9b

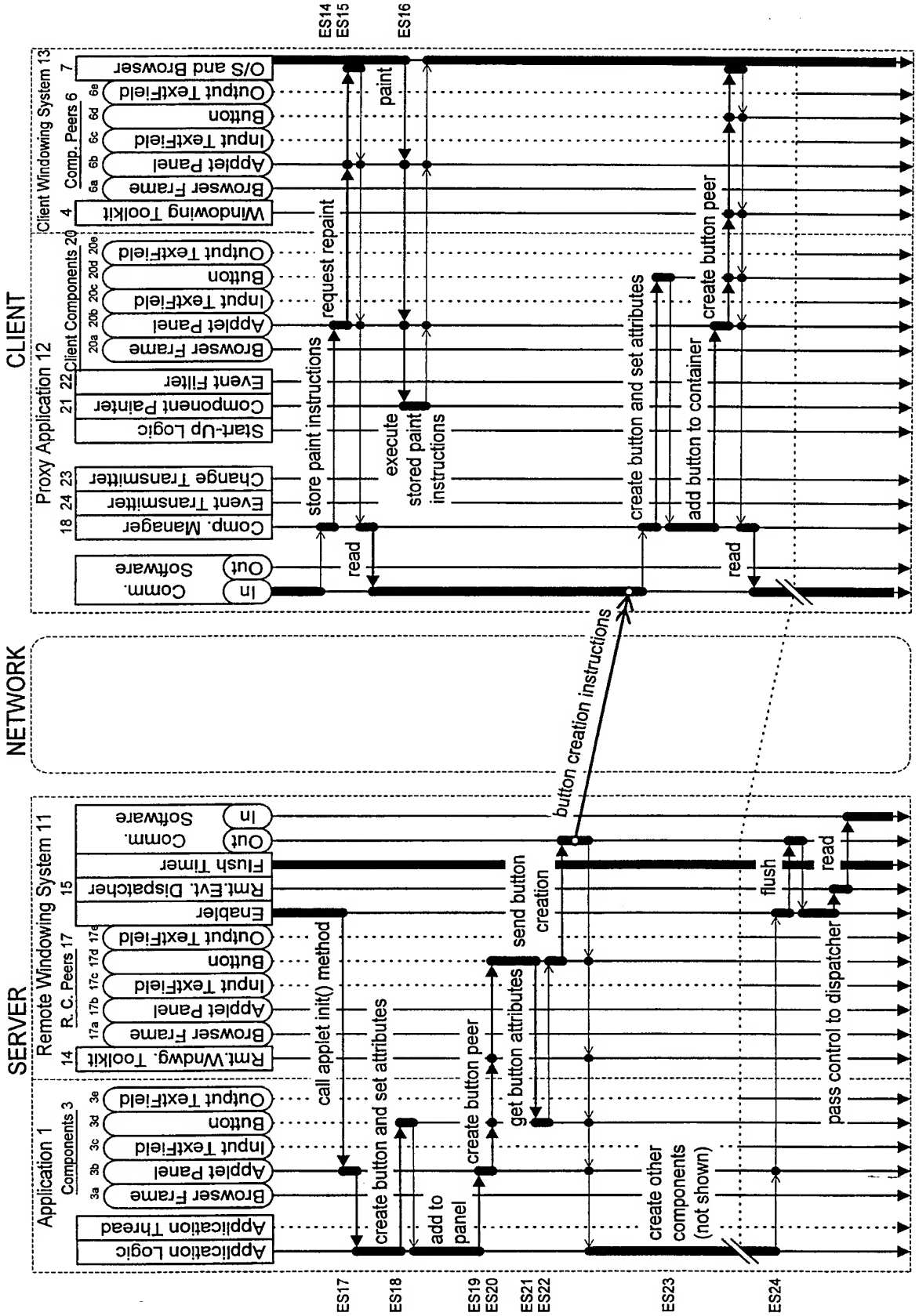

CLIENT
NETWORK
SERVER
Client Windowing System 13
Comp. Peers 6
O/S and Browser
Output TextField
Button
Input TextField
Applet Panel
Browser Frame
Windowing Toolkit
Proxy Application 12
Client Components 20
Output TextField
Button
Input TextField
Applet Panel
Browser Frame
Event Filter
Component Painter
Start-Up Logic
Change Transmitter
Event Transmitter
Comp. Manager
Comm. Software
Remote Windowing System 11
R. C. Peers 17
Output TextField
Button
Input TextField
Applet Panel
Browser Frame
Rmt.Wndwg. Toolkit
Enabler
Rmt.Evt. Dispatcher
Flush Timer
Comm. Software
Application 1
Components 3
Output TextField
Button
Input TextField
Applet Panel
Browser Frame
Application Logic
Application Thread
ES14
ES15
ES16
ES17
ES18
ES19
ES20
ES21
ES22
ES23
ES24
paint
request repaint
store paint instructions
read
execute stored paint instructions
create button and set attributes
add button to container
create button peer
read
button creation instructions
call applet init() method
create button and set attributes
add to panel
create button peer
get button attributes
send button creation
create other components (not shown)
flush
read
pass control to dispatcher

Figure 9c

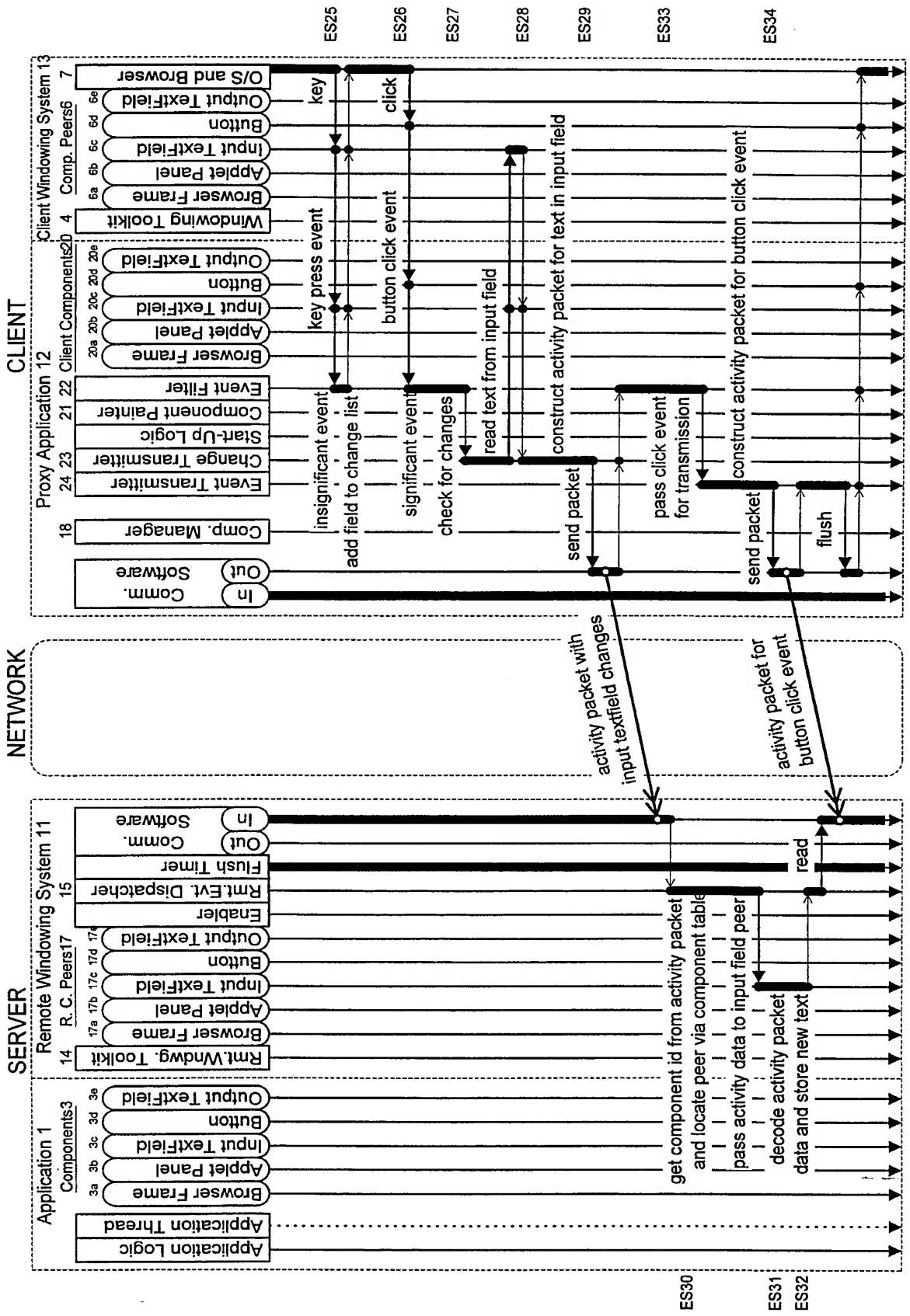

CLIENT
NETWORK
SERVER
Client Windowing System 13
Comp. Peers6
O/S and Browser
Output TextField
Button
Input TextField
Applet Panel
Browser Frame
Windowing Toolkit
Proxy Application 12
Client Components20
Event Transmitter
Change Transmitter
Start-Up Logic
Component Painter
Event Filter
Comp. Manager
Comm. Software
Application 1
Components3
Application Logic
Application Thread
Remote Windowing System 11
R. C. Peers17
Rmt.Wndwg. Toolkit
Rmt.Evt. Dispatcher
Flush Timer
Enabler
key
click
key press event
button click event
insignificant event
add field to change list
significant event
check for changes
read text from input field
construct activity packet for text in input field
send packet
pass click event for transmission
construct activity packet for button click event
send packet
flush
activity packet with input textfield changes
activity packet for button click event
get component id from activity packet and locate peer via component table
pass activity data to input field peer
decode activity packet data and store new text
read
ES25
ES26
ES27
ES28
ES29
ES33
ES34
ES30
ES31
ES32

Figure 9d

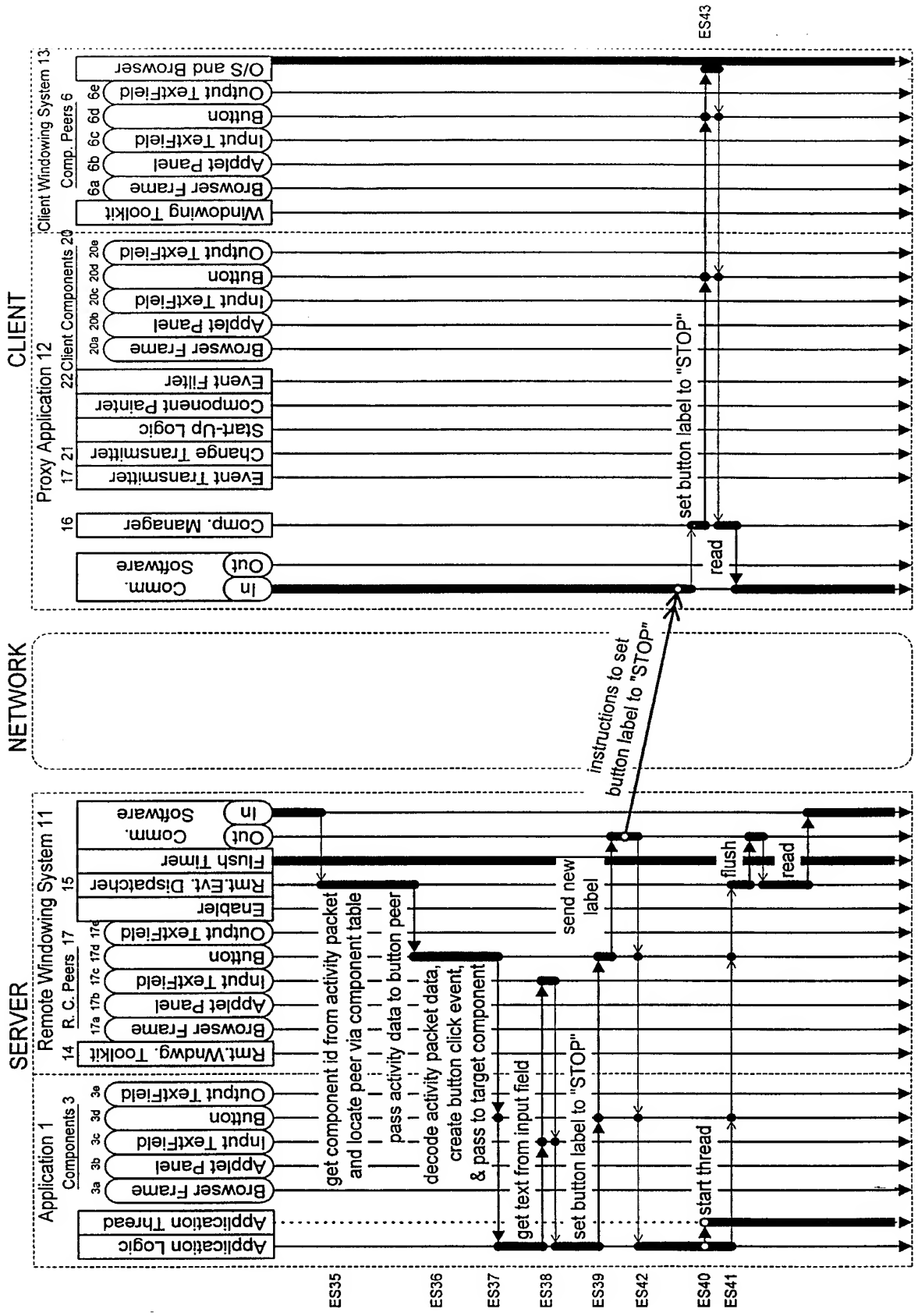

SERVER
CLIENT
NETWORK
Application 1
Components 3
Application Logic
Application Thread
Browser Frame
Applet Panel
Input TextField
Button
Output TextField
Remote Windowing System 11
R. C. Peers 17
Rmt.Wndwg. Toolkit
Rmt.Evt. Dispatcher
Enabler
Flush Timer
Comm. Out
Comm. In
Proxy Application 12
Client Components 20
Event Transmitter
Change Transmitter
Start-Up Logic
Component Painter
Event Filter
Comp. Manager
Client Windowing System 13
Comp. Peers 6
Windowing Toolkit
O/S and Browser
get component id from activity packet and locate peer via component table
pass activity data to button peer
decode activity packet data, create button click event, & pass to target component
get text from input field
send new label
set button label to "STOP"
start thread
flush
read
instructions to set button label to "STOP"
set button label to "STOP"
read
ES35
ES36
ES37
ES38
ES39
ES42
ES40
ES41
ES43

Figure 9e

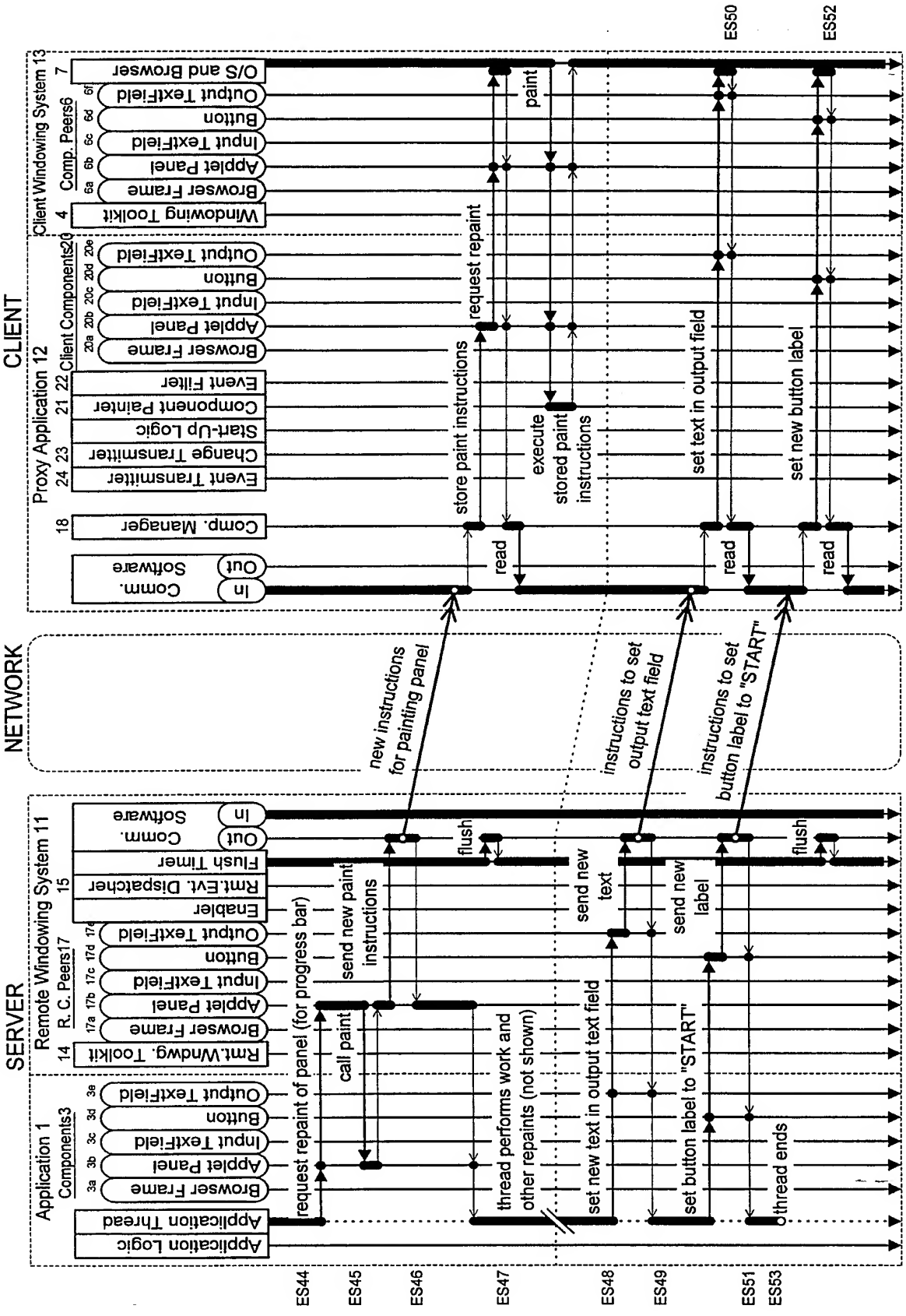

SERVER
CLIENT
NETWORK
Application 1
Components3
Application Logic
Application Thread
Browser Frame
Applet Panel
Input TextField
Button
Output TextField
Remote Windowing System 11
R. C. Peers17
Rmt.Wndwg. Toolkit
Enabler
Rmt.Evt. Dispatcher
Flush Timer
Comm. Software
Proxy Application 12
Client Components20
Event Transmitter
Change Transmitter
Start-Up Logic
Component Painter
Event Filter
Comp. Manager
Client Windowing System 13
Windowing Toolkit
Comp. Peers6
O/S and Browser
request repaint of panel (for progress bar)
call paint
send new paint instructions
flush
thread performs work and other repaints (not shown)
store paint instructions
request repaint
read
paint
execute stored paint instructions
new instructions for painting panel
set new text in output text field
send new text
instructions to set output text field
set button label to "START"
send new label
set text in output field
instructions to set button label to "START"
set new button label
thread ends
ES44
ES45
ES46
ES47
ES48
ES49
ES50
ES51
ES52
ES53

Figure 9f